View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Digital library of Brno University of Technology

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNíHO INŽENÝRSTVí
ÚSTAV MATEMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF MATHEMATICS

TRAFFIC ASSIGNMENT OPTIMIZATION MODELS

MODELY OPTIMALIZACE DOPRAVY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN HOLEŠOVSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. PAVEL POPELA, Ph.D.

BRNO 2012

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav matematiky
Akademický rok: 2011/2012

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Jan Holešovský

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Matematické inženýrství (3901T021)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Modely optimalizace dopravy

v anglickém jazyce:

Traffic assignment optimization models

Stručná charakteristika problematiky úkolu:

Student se seznámí s problematikou úloh dvoustupňového a dvouúrovňového programování včetně úloh stochastické optimalizace. Zvláštní pozornost bude věnovat dopravním úlohám a přístupům podobným úloze "network interdiction". Bude se zabývat modifikací modelů, studiem jejich vlastností, jejich transformacemi a algoritmy jejich řešení.

Cíle diplomové práce:

Modifikace a řešení vybraných dopravních problémů pomocí modelů matematické optimalizace, studium jejich vlastností a jejich aplikace na "traffic assignment problem" (TAP).

Seznam odborné literatury:

Shapiro, A. et al.: Handbook of Stochastic Programming, Elsevier 2002.
Kall, P., Wallace, S. W.: Stochastic Programming, Wiley, 1994.
Beckmann, M., et al.: Studies in the Economics of Transportation, New Haven 1956.
Patriksson, M.: The Traffic Assignment Problem - Models and Methods, VSP 1994.

Vedoucí diplomové práce: RNDr. Pavel Popela, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2011/2012.

V Brně, dne 26.10.2010

L.S.

prof. RNDr. Josef Šlapal, CSc. Ředitel ústavu	prof. RNDr. Miroslav Doupovec, CSc. Děkan fakulty

Abstrakt
Optimalizace toku v síti je klasickou aplikací matematického programování. Tyto modely mají, mimo jiné, široké uplatnění také v logistice, kde se tak snažíme docílit optimálního rozdělení dopravy, např. vzhledem k maximalizaci zisku, či minimalizaci nákladů. Toto pojetí ovšem často problém idealizuje, poněvadž předpokládá existenci jediného rozhodovatele. Takový přístup je možný ve striktně organizovaných sítích jako např. v logistických sítích přepravních společností, železničních sítích či armádním zásobování. Úloha "Traffic Assignment Problem" (TAP) se zaměřuje na dopady teorie her na optimalizaci toku, tj. zkoumá vliv více rozhodovatelů na celkové využití sítě. V práci se zaobíráme úlohou TAP s působením náhodných vlivů, k čemuž využíváme metod stochastické a vícestupňové optimalizace. Dále zkoumáme možnosti zlepšení stávajícího využití sítě za rozhodnutí autoritativního rozhodovatele, kterému je umožněn zásah do samotné struktury sítě, k čemuž využíváme víceúrovňové programování.

Summary
The class of network flow problems is one of the traditional applications of mathematical optimization. Such problems are widely applicable for example in logistics to achieve an optimal distribution of flow with respect to maximization of profit, or minimization of costs. This approach often leads to simplified models of real problems as it supposes the existence of only one decision maker. Such approach is possible in centralised networks, where an authority exists (such as railway network, military supply, or logistic network used by any company). The Traffic Assignment Problem (TAP) deals with impact of game theory to the network flow problem. Hence, we assume multiple decision makers, where each one of them wants to find his optimal behaviour. In this thesis, we focus on stochastic influences in TAP, for which we use methods of stochastic and multi-stage programming. Further, we concentrate on improvement options for the utilization of the system. Hereby, we consider possible actions of the master decision maker, and discuss them by the presence of multi-level mathematical programming.

Klíčová slova
dopravní problém, toky v sítích, stochastické programování, dvoustupňová optimalizace, dvouúrovňová optimalizace

Keywords
traffic assignment, network flows, stochastic programming, two-stage optimization, bilevel optimization

HOLEŠOVSKÝ, J. *Traffic assignment optimization models.* Brno: Brno University of Technology, Faculty of Mechanical Engineering, 2012. 82 s. Supervisor RNDr. Pavel Popela, Ph.D.

I declare that I have written the master thesis *Traffic assignment optimization models* on my own according to the instruction of my supervisor RNDr. Pavel Popela, Ph.D., and using the sources listed in references.

May 15, 2012 Bc. Jan Holešovský

I would like to thank my supervisor RNDr. Pavel Popela, Ph.D., for valuable ideas on my work, for his comments and suggestions on improving my thesis.

Bc. Jan Holešovský

Contents

1. Introduction

Network flow problems are widely studied patterns that have numbers of applications in real life. For many of these problems, useful algorithms are known that guarantee different accuracy. Recall some of them: Shortest Path Problem, Travelling Salesman Problem, Maximal Flow, or Minimal Cost Network Flow. In this thesis, the last one and its derivatives are our objectives to study.

In the first chapter (2), we define basic notions from graph theory which are further used in the thesis. Hereby, we standardize our understanding of a network, and state assumptions that follow us further in other parts on. Moreover, we try to pull the reader in the rigorousness of our conventions to avoid later misunderstandings. An important concepts are primarily definition of flow, elements of graph such as path and chain, or incidence matrix.

A short introduction to mathematical programming and its properties is made in the next chapter 3. These results are widely used in following parts in order to find a solution of a problem, or to discuss significant attributes which lead to simplification. To preface Minimal Cost Network Flow problem (MCF) closely solved in chapter 4, we show basic ideas of linear mathematical programming (LP). Due to application of MCF in logistics, we present some results linked with integral conditions. Last part of chapter 3 precedes derivatives of MCF, which we focus on immediately.

A minimal cost network flow is task to find, for a given amount of goods, the cheapest way to traverse through a graph. Typically, since each undirected or mixed graph can be transformed into directed one, we consider digraphs. The use of the shortest path is limited, as in praxis, by capacity constraints. This assumption comes true especially in logistic networks, where capacity plays a significant role. This can be caused by the nature of network, like transport over sea or usage of ferries. Some problems can rise up within the high volume of transported goods, etc. Introduction into network problematic focused on MCF is made in the third chapter. As a generalisation, we consider MCF with multiple "commodities" (so-called Multicommodity Flow), i.e. with multiple sort of goods.

Traffic Assignment Problem (TAP), the aim of our thesis, is presented right after. It is a variation of MCF, or rather of Multicommodity Flow problem (MF), where we suppose more decision makers. It is important to highlight, that in a common MCF we consider only one decision maker. This can be a boss of a logistic company, commander responsible for military supply, railway schedule planner, etc. The question being opened in TAP is as follows: What if there are more decision makers? Moreover, what if each peace of transported goods can make its own decision? Although this seem to be awesome image, most of us meet this problem every day on the streets. Here, each driver is considered as rational decision maker, and in collaboration with game theory are defined two main perspectives: system-optimal point of view, and a group of uncoordinated users.

Moreover, in TAP we suppose cost of flow as time needed to traverse a path, which, in order to simulate the real traffic in a network, depends on the flow volume. Such reasonable requirement makes problem much more complex and leads to mixed integer nonlinear programming. Since more decision makers negatively influence the solution of minimal cost, we discuss possible bound on this degradation.

Chapter 5 is focused on various changes in TAP, especially effect of randomness in cost of flow or number of users in the network. Here, we use results observed from stochastic programming, and solve TAP by building deterministic reformulations of problems. We can, for example, express a stochastic problem as a two-stage programming problem with recourse. All reached knowledges are tested on simple network, for which we used software GAMS. GAMS is powerful optimization tool that allows us to solve such complicated problems as presented.

Network Design is the aim of the last part of the thesis, the chapter 6. We deal with task to decide about structure of the network in order to improve the utilization of the system. Braess in his article [5] first presented existence of so-called Braess paradox, that may occur with adding a new arc in the network. This paradox may occur in any network, and it negatively influences the utilization and the cost of a solution. Discussion is made, where we think about limitations or improving algorithms of the Braess paradox. Prove that such thing may appear in praxis, is represented by The New York Times article [26].

In the last sections, we transform the Braess paradox problematic into a new problem called *Street Cleaning Problem* (SCP). This is a variation of typical Network Design task. The objective of SCP is to decide about the best sequence of roads that need to be closed. We show solution of this problem which is presented as bilevel program. After consideration of randomness in the volume of drivers, we get various deterministic reformulations of SCP, for example SCP as two-stage bilevel problem with recourse.

All discussed problems were implemented in software GAMS, and we present our results in corresponding parts of the text. To the thesis are enclosed two main parts at the end. One treating with the GAMS programming language and a structure of problem solvable by GAMS. The latter presents one of the most important theorems from the game theory. This, along with chapter 4, gives the reader closer insight to the relation between TAP and existence of an equilibrium.

2. Basics of the Graph Theory

In this chapter, we shortly describe graph in the meaning of graph theory, define some basic concepts and show important properties of graphs. As we will see in the following chapters, many applications of mathematical programming are based on the graph theory. We assume, that the reader has already met some parts of the theory and therefore some concepts are introduced in comments instead of definitions, moreover some of them that are not further needed are skipped.

Our short resume is primarily based on [17] and on books written by *Bazaraa*, *Jarvis* [2] and *Dantzig*, *Thapa* [9].

2.1. Definition of a Graph

A graph, in the meaning of the graph theory, is an object consisting of *nodes* (or *vertices*) and links between them, so called *edges* (or *arcs*).

Definition 2.1 (Simple undirected graph).
A *simple undirected graph* (also an *ordinary graph* or a *simple graph*) is a pair $G = (N, E)$ where N is a finite set of *nodes* and $E = \{\{u, v\} | u, v \in N, u \neq v\}$ is a finite set of *(undirected) edges*. We say that an edge $e = \{u, v\}$ is *incident* on nodes u and v, or that nodes u and v are incident on edge e.

Definition 2.2 (Simple directed graph).
A *simple directed graph* (or *simple digraph*) is a pair $G = (N, A)$ where N is a finite set of nodes and $A = \{(u, v) | u, v \in N, u \neq v\}$ is a finite set of *directed edges* (or *arcs*). We say that an arc $a = (u, v)$ is *incident* on nodes u and v, or that nodes u and v are incident on arc a. Often, we say that u is the *tail* and v is the *head*.

As we can see, an edge is a link between two nodes, while an arc is a directed link. Thus, arc may represent an flow in the graph, or better in the network. [1] Graphs are commonly drawn in graphical way: nodes are displayed as points or circles, linked by edges (lines) or arcs (arrows directed from tail to head node). We can also say that an arc (u, v) connects u to v, and that an edge connects u and v.

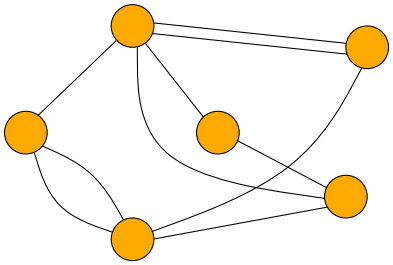
There can be found an definition of *mixed graph* in some books, i.e. graph containing both edges and arc. Graph of this type can be usually transformed by edges directing or arcs symmetrizing to directed or undirected graph. More often are mixed graphs considered as a little bit more complicated directed graphs, and this is exactly the way we will understand these graphs.

Definition 2.3 (Multigraph or Graph).
Multigraph (or *graph*) is a triple $G = (N, E, \epsilon)$ where N is a finite set of nodes, E is a finite set of edges and ϵ is mapping assigning to each edge one- or two-element set of nodes.

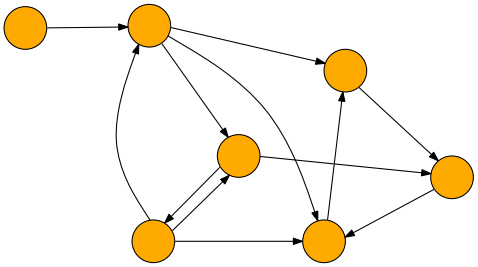
[1] In some books, the term graph and network are used as synonyms, while sometime the meanings are different. We will define the term of network later.

Definition 2.4 (Directed graph).
Directed graph (or *digraph*) is a triple $G = (N, A, \epsilon)$ where N is a finite set of nodes, A is a finite set of arcs and $\epsilon : A \to N \times N$ is a mapping assigning to each arc an ordered pair of nodes.

Multigraph or digraph, are just generalization of simple graphs where two nodes are connected with at most one link (edge or arc). In a multigraph or a digraph, the number of links is arbitrary. Moreover, no one of all definition restricts so called *self-loops*, which are sometimes not taken into account.



A graph



A digraph

Definition 2.5 (Subgraph).
If $G' = (N', E')$ and $G = (N, E)$ are ordinary graphs, we say that G' is *subgraph* of G, if $N' \subseteq N$ and $E' \subseteq E$. Moreover, G' is called *subgraph of G induced by its nodes*, if

$$(u, v \in N', \{u, v\} \in E') \Rightarrow \{u, v\} \in E.$$

Definition 2.6 (Node degree).
Let $G = (N, E, \epsilon)$ be a graph and $u \in N$ a node. We define the *degree of the node u* as integer number

$$\deg(u) = |\{e \in E | \epsilon(e) = \{u, v\}, v \in N\}|.$$

Definition 2.7.
Let $G = (N, A, \epsilon)$ be a digraph. For a node $u \in N$ we define the *degree of the node u* as integer number

$$\deg(u) = |\{a \in A | \epsilon(a) = (u, v) \wedge \epsilon(a) = (v, u), v \in N\}|.$$

Further, we define number of arcs leading from node u as the *indegree* $\deg_+(u) = |\{a \in A | \epsilon(a) = (u, v), v \in N\}|$, and number of arcs that leading to node u as the *outdegree* $\deg_-(u) = |\{a \in A | \epsilon(a) = (v, u), v \in N\}|$. Thus

$$\deg(u) = \deg_+(u) + \deg_-(u).$$

For both, graph and digraph, the degree gives the number of links incident on a specific node. In the case of digraph, we can divide the set of incident arcs into two parts: directed in and directed out of node.

Definition 2.8 (Bipartite graph).
A graph $G = (N, E)$ is called *bipartite graph* if its nodes can be divided in two disjoint sets N_1, N_2, such that this relation holds: $\{u, v\} \in E \Rightarrow u \in N_1, v \in N_2$.

It is easy to see, that a graph G is bipartite if and only if G contains no circles of odd length.

2.2. Walk, Path and Simple Path

Definition 2.9 (Walk).
In a graph $G = (N, E, \epsilon)$ we define a *walk* between nodes $u, v \in N$ of length n as a sequence $(u = w_0, e_1, w_1, e_2, \ldots, e_n, w_n = v)$, such that for all $i = 0, 1, \ldots, n$ and all $j = 1, 2, \ldots, n$, following properties hold

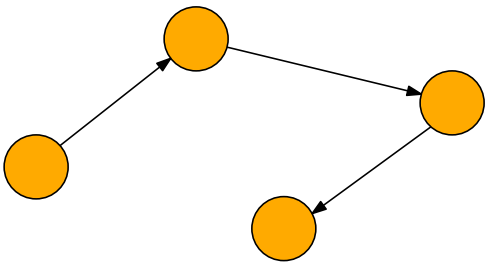
$$w_i \in N, e_j \in E, \epsilon(e_j) = \{w_{j-1}, w_j\}.$$

Thus in a walk the nodes and edges may repeat.

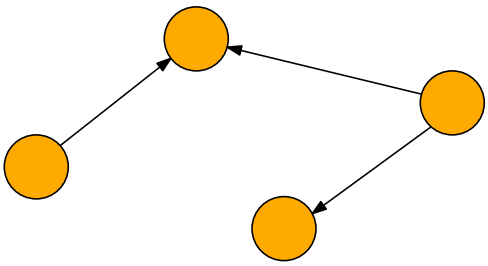
Definition 2.10 (Path).
In a graph $G = (N, E, \epsilon)$ we define a *path* between nodes $u, v \in N$ of length n as a walk $(u = w_0, e_1, w_1, e_2, \ldots, e_n, w_n = v)$, such that $i \neq j \Rightarrow e_i \neq e_j$, $1 \leq i, j \leq n$.

Definition 2.11 (Simple path).
In a graph $G = (N, E, \epsilon)$ we define a *simple path* between nodes $u, v \in N$ of length n as a walk $(u = w_0, e_1, w_1, e_2, \ldots, e_n, w_n = v)$, such that $i \neq j \Rightarrow w_i \neq w_j$, $0 \leq i, j \leq n$.

Definition 2.12 (Path and chain in a digraph).
In a digraph, we distinguish two cases: a *path* is a sequence of arcs $((w_0, w_1), (w_1, w_2), \ldots, (w_{n-1}, w_n))$, in which the initial node of each arc is the same as the terminal node of the preceding arc in the sequence (thus all arcs are directed from w_0 toward w_n). A *chain* is similar to a path, except that not all arcs need to be directed toward w_n.



A path in a digraph



A chain in a digraph

In a path nodes may repeat, but all edges must be different. Simple path is a walk, where nodes may not repeat. Obviously, in graph there is a simple path between u and v if there is a walk between u and v. In a similar way we may define a simple path or simple chain in a digraph.

Definition 2.13 (Circle, Cycle).
Let $G = (N, E, \epsilon)$ be a graph. A walk $(u = w_0, e_1, w_1, e_2, \ldots, e_n, w_n = v)$, such that $i \neq j \Rightarrow w_i \neq w_j$, $0 \leq i, j \leq n-1, w_0 = w_n$, is called a *circle*.

Circle is thus a walk in which all the nodes are different except the first and the last one. Directed circle is called a *cycle*.

Definition 2.14 (Connected graph).
A graph $G = (N, E, \epsilon)$ is called *connected graph* if for any pair of nodes $u, v \in N$ there is a walk between u and v.

Definition 2.15 (Connected digraph).
A digraph $G = (N, A)$ is said to be *(weekly) connected digraph* if for any pair of nodes $u, v \in N$ there exists in G a chain from u to v. Further, a graph is said to be *strongly connected digraph* if for every pair $u, v \in N$ there exists a path between u and v.

Definition 2.16 (Components).
For graphs $G' = (N', E', \epsilon)$ and $G = (N, E, \epsilon)$ we say that G' is a *component of* G, if G' is connected subgraph of G induced by its nodes, and any subgraph $G'' = (N'', E'', \epsilon)$ of G, such that $N' \subset N'', E' \subset E''$, is disconnected.

Definition 2.17 (Tree).
Tree is a connected graph with no circles.

Thus, in a tree there is a unique simple path between every pair of nodes.

Definition 2.18 (Spanning tree).
Let $G = (N, E)$ be a ordinary graph. Subgraph $T = (N', E')$ of G is called a *spanning tree of* G, if T is a tree and $N' = N$.

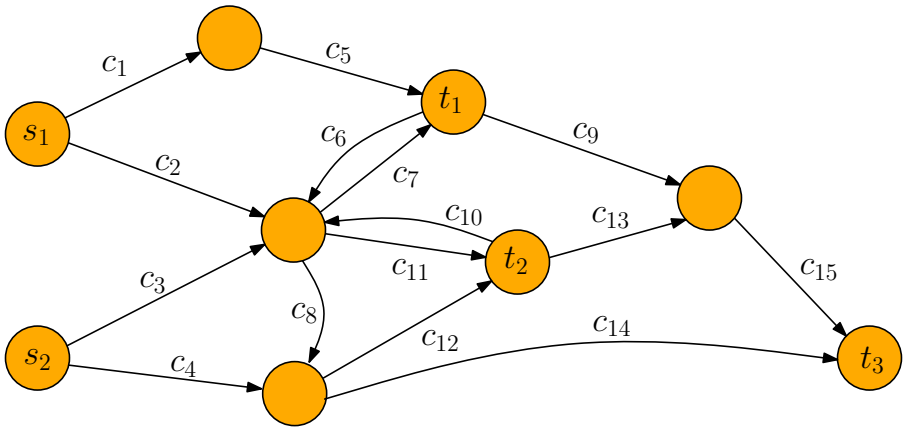
2.3. Weighted Graph and Network

Definition 2.19 (Weighted graph).
Let $G = (N, E)$ be a simple graph. If a mapping $w : E \to \mathbb{R}$ is given, then the triple $G = (N, E, w)$ is called a *weighted graph*. Each edge $e \in E$ is assigned a real number $w(e)$, called the *weight of edge* e or *length of edge* e.

For a subgraph $G' = (N', E')$ of G, we define the *weight of* G' as number $w(G') = \sum_{e \in E'} w(e)$.

Definition 2.20 (Network).
A *network* is quadruple $\mathcal{N} = (G, N_s, N_t, c)$ where $G = (N, A)$ is a simple digraph, $N_s, N_t \subset N, N_s \cap N_t = \emptyset$, and $c = c((u, v)) \geq 0 \quad \forall (u, v) \in A$ is *capacity*. If $(u, v) \notin A$, we put $c((u, v)) = 0$. Set N_s is called *set of sources* and N_t is called *set of targets*.

We can imagine network as a pipeline starting with node $s \in N_s$ and ending with node $t \in N_t$.



NETWORK WITH TWO SOURCES AND THREE TARGETS.

DEFINITION 2.21 (FLOW).
Let $G = (N, A)$ be a simple digraph and $\mathcal{N} = (G, N_s, N_t, c)$ be a network. *Flow in a network* $\mathcal{N}$ is a mapping $f : N \times N \to \mathbb{R}$, such that

a) $f(u, v) \leq c(u, v) \qquad \forall u, v \in N,$

b) $f(u, v) = -f(v, u) \qquad \forall u, v \in N,$

c) $\sum_{v \in N} f(u, v) = 0 \qquad \forall u \in N - (N_s \cup N_t).$

2.4. Incidence Matrix of a Graph

A general graph may be represented in different ways including the enumeration of nodes and edges. A convenient way to describe a general graph is to use so called *incidence matrix*. An incidence matrix is zero-matrix containing nonzero elements reflecting the relations between nodes and arcs, or between two nodes in such a graph. For a directed graph typically $+1$ and -1 elements are used. In a case of a multigraph, these elements are ones (showing that nodes are incident on an edge).

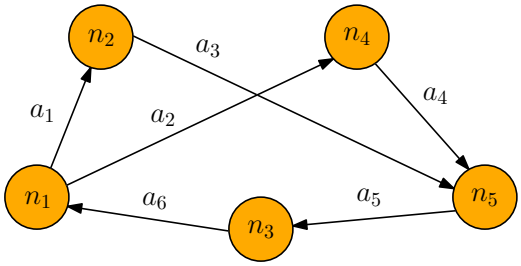
In this section, we describe only incidence matrices for a digraph, since such a matrix for a multigraph contains ones at the same places where the previous one contains $+1$s and -1s (recall that multigraph can be interpreted as digraph replacing an edge by two arcs).

Node-arc incidence matrix. Let $G = (N, A)$ be a simple digraph. The *node-arc incidence matrix* $\mathbf{A}$ of graph G is $m \times n$ matrix, such that $m = |N|$ and $n = |A|$. In $\mathbf{A}$ the nodes correspond to the rows and the arcs correspond to the columns. Since each arc has a tail and a head, each column of $\mathbf{A}$ contains exactly two non-zero coefficients. A column associated with arc $(i, j) \in A$ contains $+1$ in row i and -1 in row j.

In the undirected case both entries are $+1$.

Node-node incidence matrix. Let $G = (N, A)$ be a simple digraph. The *node-node incidence matrix* $\mathbf{A}$ of graph G is a $m \times m$ matrix, such that $m = |N|$. Each node corresponds to each row and column. The only non-zero entries of matrix $\mathbf{A}$ are $a_{ij} = 1$ and $a_{ji} = -1$ for each arc $(i, j) \in A$.

In the undirected case both entries are +1 and matrix **A** is symmetric.



$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 1 & 0 \end{bmatrix}$$

A NODE-ARC INCIDENCE MATRIX OF A GRAPH

In the following text, we understand under *incidence matrix* the above described node-arc incidence matrix.

LEMMA 2.1 (RANK OF THE INCIDENCE MATRIX).
The node-arc incidence matrix **A** of a connected digraph with $m \geq 2$ nodes has rank $r(\mathbf{A}) = m - 1$.

Proof of this useful lemma is shown in [2] or in [9]. Moreover, this result is valid also for undirected graphs, since all graphs can be represented as "more complicated" digraphs. Thus, connected graph can be in the light of definition 2.15 understood as connected digraph.

3. Mathematical Programming

Mathematical programming, or optimization, is concerned with finding an optimal solution of given problem, e.g. finding optimal value of a function, with respect to constraints, that must be satisfied. These constraints occur as inequalities or equalities. If we suppose $X \subset \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, then mathematical program (MP) in its "rough form" can be written as follows:

$$\text{Minimize (or maximize) } f(\mathbf{x}) \text{ subject to } \mathbf{x} \in X.$$

More precisely, as an MP we understand a deterministic MP, where both the function f and the set X are known, and do not depend on any unknown parameter (like random variables, etc.). The task of MP is not only to find the minimal value of function f, denoted $f^* = \min_{\mathbf{X}} f(\mathbf{x})$, but also to find the set of points $\mathbf{x} \in \mathbb{R}^n$, in which function f takes the optimal value. This set is called *set of optimal solutions*, and will be denoted X^* or $\text{argmin}_{\mathbf{X}}\{f(\mathbf{x})|\mathbf{x} \in X\}$.

There are various special cases of mathematical programming, much easier to study than MPs in general form. Depending on the properties of function to minimize (or maximize) and set X, we get:

- *Linear programming*, if the set X is convex polyhedral and function f is linear.
- *Nonlinear programming*, if either function f is nonlinear, or set X is not polyhedric:
  - *convex:* set X and function f remain convex. For this case, we may use many reached results.
  - *nonconvex:* either set X, or function f are not convex.
- *Integer and mixed integer programming*, if set $X \subset \mathbb{Z}^n$, or $X \subset \mathbb{Z}^k \times \mathbb{R}^{n-k}$, respectively.
- *Binary programming*, if set $X \subset \{0, 1\}^n$.

In practical problems, all types of MP are mixed, e.g. we should solve linear program with integer and binary variables. Then we must combine results reached in all branches of mathematical programming. The subsequent section is focused on linear programming. Hereafter, we use abbreviation *min* (*max*) instead of *minimize* (*maximize* respectively), and *s.t.* instead of *subject to*.

3.1. Linear Programming

A linear program (LP) is a problem of minimizing or maximizing a linear function in the presence of linear constraints of the inequality and/or the equality type.

LP in basic notation. Consider the following linear programming problem:

$$\begin{array}{ll} \min & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{s. t.} & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \quad \geq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \quad \geq b_2 \\ & \qquad\qquad \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \quad \geq b_m \\ & x_1, x_2, \ldots, x_n \geq 0 \end{array} \tag{3.1}$$

The function $z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$ is the *objective function* to be minimized. The coefficients $c_1, c_2, \ldots, c_n$ are known (they are sometimes called *cost coefficients*), and $x_1, x_2, \ldots, x_n$ are the *variables* (or *decision variables*) to be determined. The inequality $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$ denotes the i-th *constraint*. Coefficients b_i on the right-hand side form the *right-hand-side vector* $\mathbf{b}$, and represent the minimal requirements to be satisfied. Coefficients a_{ij} form the *constraint matrix* $\mathbf{A}$ given below.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}.$$

The constraints $x_1, x_2, \ldots, x_n \geq 0$ are the *nonnegativity constraints*, and they often occur in mathematical programs.

A sequence of variables $x_1, x_2, \ldots, x_n$ satisfying all the constraints is called a *feasible vector* (or a *feasible solution*, a *feasible point*). The set of all these feasible vectors form *feasible region* (or *feasible space*).

LP in matrix notation. We will denote a LP in matrix form, which is much more convenient. Denote $\mathbf{o} = (0, 0, \ldots, 0)^T$ the zero vector of an appropriate length, and variables and coefficients as follows:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}.$$

Then the LP problem (3.1) can be written[1] as

$$\begin{array}{ll} \min & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{o}. \end{array} \tag{3.2}$$

A feasible vector $\mathbf{x}^*$ is called *optimal solution* of problem (3.1, 3.2), if for all feasible vectors $\mathbf{x}$ in (3.1, 3.2) is satisfied $f(\mathbf{x}) \geq f(\mathbf{x}^*)$. For maximization problem the optimality condition is $f(\mathbf{x}) \leq f(\mathbf{x}^*)$. The optimal value of objective function is denoted $z^* := z(\mathbf{x}^*)$.

[1] Let us understand the inequality $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ as inequalities by its components.

Inequalities and equations. An inequality can be easily transformed into an equation. Consider the constraint given by $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$. This constraint can be conversed into an equation by subtracting a nonnegative *slack variable* x_{n+i} as follows:

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i \quad \longmapsto \quad \sum_{j=1}^{n} a_{ij}x_j - x_{n+i} = b_i,$$
$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \quad \longmapsto \quad \sum_{j=1}^{n} a_{ij}x_j + x_{n+i} = b_i.$$

Conversely, an equation $\sum_{j=1}^{n} a_{ij}x_j = b_i$ can be transformed into two inequalities $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$ and $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$.

Nonnegativity of the variables. Mostly, we need to express variables in "known" form of nonnegativity constraints, although they may take any value. Bounded variables we may transform as follows:

$$x \geq l \quad \longmapsto \quad x' = x - l \geq 0,$$
$$x \leq u \quad \longmapsto \quad x' = u - x \geq 0.$$

If variable x is unrestricted ($x \in \mathbb{R}$), we may replace it by $x = x' - x''$ where $x', x'' \geq 0$.

Minimization and maximization problems. Easy conversion between minimization and maximization problems can be provided in following way:

$$\min \mathbf{c}^T\mathbf{x} = -\max(-\mathbf{c}^T\mathbf{x}).$$

Since, as we can see above, any LP can be rewritten in many equivalent ways, this might be confusing. In particular, two forms are used: the standard and the canonical form. A linear program is said to be in *standard form* if all constraints are equalities and all variables are nonnegative. This form is used for traditional solution method, the simplex method, that is designed for LPs in such form.

A minimization problem is said to be in *canonical form* if all variables are nonnegative and all constraints are of the type $\geq$. A maximization problem is in canonical form if all the variables are nonnegative and all the constraint are of the $\leq$ type. All transformations are summarised in Table 1. below.

	Minimization	Maximization
Standard form	$\min \ \mathbf{c}^T\mathbf{x}$ s.t. $\mathbf{Ax} = \mathbf{b}$ $\mathbf{x} \geq \mathbf{o}$	$\max \ \mathbf{c}^T\mathbf{x}$ s.t. $\mathbf{Ax} = \mathbf{b}$ $\mathbf{x} \geq \mathbf{o}$
Canonical form	$\min \ \mathbf{c}^T\mathbf{x}$ s.t. $\mathbf{Ax} \geq \mathbf{b}$ $\mathbf{x} \geq \mathbf{o}$	$\max \ \mathbf{c}^T\mathbf{x}$ s.t. $\mathbf{Ax} \leq \mathbf{b}$ $\mathbf{x} \geq \mathbf{o}$

TABLE 1.

Properties of Linear Programs:

Hereby, we present some results from linear programming. These significant properties of LPs are based on fundamentals of convex analysis and features of convex functions. For this reason, we do not prove theorems and lemmas stated below, and we just refer to [3], [2], [9] and [20].

Definition 3.1 (Convex set).
Let $S \subset \mathbb{R}^n$. We call S a *convex set* if $\forall \mathbf{x}_1, \mathbf{x}_2 \in S$ and $\forall \lambda \in [0,1]$ is $\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in S$.

Definition 3.2 (Convex combination).
Let $\mathbf{x}, \mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{R}^n$. Then $\mathbf{x}$ is called a *convex combination of* $\mathbf{x}_1, \ldots, \mathbf{x}_k$, if it satisfies $\mathbf{x} = \sum_{j=1}^{k} \lambda_j \mathbf{x}_j$ where $\sum_{j=1}^{k} \lambda_j = 1$ and $\forall j \in \{1, \ldots, k\}$ $\lambda_j \geq 0$.

Definition 3.3 (Convex hull).
Let $S \in \mathbb{R}^n$. We say that $\mathbf{x} \in \mathbb{R}^n$ belongs to a *convex hull of* S, denoted $\mathbf{x} \in \text{conv}S$, if $\exists k \in \mathbb{N}, \exists \lambda_1, \ldots, \lambda_k \geq 0, \sum_{j=1}^{k} \lambda_j = 1$ $\exists \mathbf{x}_1, \ldots, \mathbf{x}_k \in S : \mathbf{x} = \sum_{j=1}^{k} \lambda_j \mathbf{x}_j$.

Convex hull of set S is the smallest convex set containing S. Moreover, it is the intersection of all convex sets containing S.

Definition 3.4 (Hyperplane, Linear halfspace).
Let $\alpha \in \mathbb{R}, \mathbf{p} \in \mathbb{R}^n$, and $\mathbf{p} \neq \mathbf{o}$. A *hyperplane* is defined by $H = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{p}^T \mathbf{x} = \alpha\}$. The hyperplane defines two *(linear) halfspaces* $H^+ = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{p}^T \mathbf{x} \circ \alpha\}$, where $\circ$ denotes a following relation: a halfspace is called *closed halfspace*, if $\circ \in \{\leq, \geq\}$, or *open halfspace*, if $\circ \in \{<, >\}$.

Lemma 3.1 (Convexity of hyperplanes and halfspaces).
Let $\mathbf{p} \in \mathbb{R}^n, \alpha \in \mathbb{R}$. Then the hyperplane $H = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{p}^T \mathbf{x} = \alpha\}$ and the halfspace $H^+ = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{p}^T \mathbf{x} \circ \alpha\}$ from the definition above are convex sets.

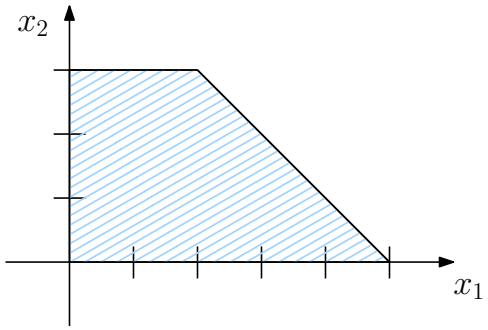
Lemma 3.2 (Intersection of convex sets).
Intersection of arbitrary many convex sets is convex set.

Definition 3.5 (Polyhedral set).
$S \in \mathbb{R}^n$ is called a *polyhedral set* if it is intersection of a finite number of closed halfspaces.

Obviously, a polyhedral set is convex set, and thus feasible region of a linear program is also convex.

Definition 3.6 (Polytope).
Let $k \in \mathbb{N}$, and let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k \in \mathbb{R}^n$. Then $\text{conv}\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ is called a *polytope*.

Thus, a polytope is polyhedral set, and a bounded polyhedral set is a polytope.



$$\begin{aligned} \min \quad & 3x_1 + 5x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq 5, \\ & x_1 \geq 0, \\ & 0 \leq x_2 \leq 3. \end{aligned}$$

Feasible region of a LP is a polyhedral set.

Definition 3.7 (Extreme point).
Let $S \subset \mathbb{R}^n, S \neq \emptyset$ be a convex set. We call $\mathbf{x} \in S$ an *extreme point* (EP) of S, if for all $\mathbf{x}_1, \mathbf{x}_2 \in S$ and for all $\lambda \in (0, 1)$ holds

$$\mathbf{x} = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \Rightarrow \mathbf{x} = \mathbf{x}_1 = \mathbf{x}_2.$$

Hence, no one extreme point can be expressed as strict convex combination of two distinct point from the set. For a polytope, all vertices are extreme points.

Theorem 3.1 (Existence of an extreme point).
Let $S = \{\mathbf{x} | \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{o}\}$ be a nonempty set, and matrix $\mathbf{A}$ has a full row rank. Then S has at least one extreme point.

Definition 3.8 (Direction).
Let $S \subset \mathbb{R}^n, S \neq \emptyset$ be a closed convex set. We call $\mathbf{d} \in \mathbb{R}^n, \mathbf{d} \neq \mathbf{o}$, a *direction*, if for all $\mathbf{x} \in S$ and for all $\lambda \geq 0$ the point $\mathbf{x} + \lambda \mathbf{d}$ belongs to set S.

Definition 3.9 (Extreme direction).
A direction $\mathbf{d}$ of S is said to be an *extreme direction* (ED) of S if it cannot be represented as a positive combination of two distinct directions of the set, i.e. for all directions $\mathbf{d}_1, \mathbf{d}_2$ of S and all $\lambda_1, \lambda_2 > 0$ holds

$$\mathbf{d} = \lambda_1 \mathbf{d}_1 + \lambda_2 \mathbf{d}_2 \Rightarrow \exists \alpha > 0 : \mathbf{d}_1 = \alpha \mathbf{d}_2.$$

Obviously, there exists no direction in a bounded set, because the direction gives "course" where all points belong to the set. It can be easily proven, that the number of extreme directions is finite.

Moreover, every point of the set S can be represented as nonnegative combination of all extreme points and extreme directions (see Representations theorem [2]). This important result is further used to prove following theorems.

Theorem 3.2 (Optimality condition).
Assume, we have a linear program in the standard form. Let S denote the set of feasible solutions, and $D = \{\mathbf{c}^T \mathbf{x} | \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{o}\}$. The set $D \neq \emptyset$ (finite solution exists) if and only if $\mathbf{c}\mathbf{d}_j \geq 0$ for all extreme directions of the set S.

Theorem 3.3.
Any solution of a linear program that is a local minimum solution is also a global minimum solution.

Prove of this theorem follows from the one presented in the section Nonlinear programming. Linear function is from definition as convex so concave, and for the local maximum holds similar theorem.

Remark. Above presented results just illustrate how important is linear programming. More about algebraic background, convexity, and feasible region in linear programming can be found in [2], [9], [20] or [3].

Solution methods. Small problems (with two variables) are solvable in graphical way. This gives an idea for solving more complex problems. In 1949 George Dantzig published *Simplex method* for solving linear programs in standard form. Although we decided not to describe this method, it is important to mention it, because Simplex method is one of traditional ways how to solve a linear program. For the algorithm see for example [9], [2] or [20].

3.2. Integer Programming

Recall the linear program (3.2). In this problem all variables are *continuous*, i.e. they may take any real value. Often this is realistic assumption in many applications where we produce divisible goods. At other times, this assumption is not allowable.

Integer programming is traditionally linked with logistics and network problems. Mostly in this branches occur integer requirements, since amount of transported units through networks are often integer numbers (e.g. chairs, barrels, people). In this case, we must consider an *Integer linear program* (ILP) as following:

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{o} \\ & \mathbf{x} \in \mathbb{Z}^n. \end{aligned} \tag{3.3}$$

If the constraints are of network nature, the integrality restrictions can be often ignored, and solving the resulting LP we get solution for previous ILP. For cases when this is not possible, we consider approaches that have been developed, because there is no unique technique for solving integer programs. For solving integer programs are usually used following methods:

i) enumeration techniques, including branch-and-bound method (B & B),

ii) cutting-plane methods.

Definition 3.10 (Integer hull of polytope).
For any polytope P we define *integer hull* P_1 *of* P as the convex hull of the integral vectors in P.

Definition 3.11 (Integral polytope).
Let P be a polytope and P_1 its integer hull. Then P is called *integral polytope*, if $P_1 = P$.

In an integral polytope are all extreme points integral.

Definition 3.12 (Unimodular matrix).
A square matrix $\mathbf{A}$ is said to be *unimodular* if $\mathbf{A}$ is integral and $\det \mathbf{A} = \pm 1$.

Definition 3.13 (Totally unimodular matrix).
A matrix $\mathbf{A}$ is said to be *totally unimodular matrix* (TUM) if each subdeterminant of $\mathbf{A}$ is $0, +1$, or -1. In particular, each entry of totally unimodular matrix is $0, +1$ or -1.

Obviously, each square totally unimodular matrix is unimodular. This relation does not hold vice versa, but following lemma shows us how to recognize a totally unimodular matrix.

Lemma 3.3 ([20]).
Total unimodularity of a matrix is preserved under following operations:

(i) permuting rows or columns,

(ii) taking the transposition,

(iii) multiplying a row or column by -1,

(iv) repeating a row or column,

(v) adding an all-zero row or column, or adding a row or column with one nonzero, being ± 1.

Lemma 3.4 ([1]).
Let $\mathbf{A}$ be a matrix with entries $0, +1$ or -1. Then $\mathbf{A}$ is TUM if it contains no more than one $+1$ and no more than one -1 in each column.

Theorem 3.4 (Bipartite graph, [20]).
Let $G = (N, E)$ be a simple graph, $m = |N|, n = |E|$, and let $\mathbf{A}$ be the $m \times n$ node-arc incidence matrix of G. Then $\mathbf{A}$ is TUM if and only if G is bipartite.

Theorem 3.5 (Digraphs).
Let $G = (N, A)$ be a digraph, $m = |N|, n = |A|$, and let $\mathbf{A}$ be the $m \times n$ node-arc incidence matrix of G. Then $\mathbf{A}$ is TUM.

Proof. Denote $\mathbf{B}_t$ a $t \times t$ square submatrix of $\mathbf{A}$. Foregoing theorem is provable by induction on t: Since $\mathbf{A}$ is a incidence matrix, the case of $t = 1$ is trivial. Let us suppose, that all submatrices $\mathbf{B}_t$ are TUM. Then for a matrix $\mathbf{B}_{t+1}$ may occur one of three possible cases:

(i) If $\mathbf{B}_{t+1}$ contains a zero column, then automatically $\det \mathbf{B}_{t+1} = 0$.

(ii) If $\mathbf{B}_{t+1}$ contains a column with exactly one nonzero element (being ± 1), then calculate $\det \mathbf{B}_{t+1}$ using this column. Since the corresponding submatrix of size $t \times t$ is TUM, easily $\det \mathbf{B}_{t+1} \in \{-1, 0, +1\}$.

(iii) If $\mathbf{B}_{t+1}$ contains only columns with exactly two nonzero elements (being $+1$ and -1), then replace the last row with sum of all rows. This adjustment does not affect the value of determinant. Now, since the last row contains only zeros, $\det \mathbf{B}_{t+1} = 0$.

□

THEOREM 3.6 ([20]).
Let $\mathbf{A}$ be an integral matrix. Then the following are equivalent:

(i) $\mathbf{A}$ is unimodular,

(ii) for each integral vector $\mathbf{b}$, the polytope $\{\mathbf{x}|\mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{o}\}$ is integral,

(iii) for each integral vector $\mathbf{c}$, the polytope $\{\mathbf{y}|\mathbf{A}^T\mathbf{y} \geq \mathbf{c}\}$ is integral.

THEOREM 3.7 (HOFFMAN-KRUSKAL THEOREM, [20]).
Let $\mathbf{A}$ be a matrix with $0, +1$ and -1 entries. Then the following are equivalent:

(i) $\mathbf{A}$ is TUM,

(ii) for each integral vector $\mathbf{b}$, the polytope $\{\mathbf{x}|\mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{o}\}$ is integral,

(iii) for all integral vectors $\mathbf{a}, \mathbf{b}, \mathbf{l}, \mathbf{u}$, the polytope $\{\mathbf{x}|\mathbf{a} \leq \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$ is integral,

(iv) for all integral vectors $\mathbf{b}, \mathbf{u}$, the polytope $\{\mathbf{x}|\mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{o} \leq \mathbf{x} \leq \mathbf{u}\}$ is integral.

3.3. Nonlinear Programming

Ahead of section 4.3, where we deal with nonlinear MPs, we shortly discuss this specific class of optimization problems. These nonlinearities may appear as in objective function as in constraints.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be the objective function, and constraints are given by function $g : \mathbb{R}^n \to \mathbb{R}$. Let $X \subseteq \mathbb{R}$ be a set, and denote $\circ \in \{\leq, =, \geq\}$. A *nonlinear (mathematical) program* (NLP) is called the following MP

$$\begin{array}{ll} \min & \mathbf{f}(\mathbf{x}) \\ \text{s.t.} & \mathbf{g}(\mathbf{x}) \circ \mathbf{o} \\ & \mathbf{x} \in X. \end{array} \tag{3.4}$$

Recall, that function $f : S \to \mathbb{R}$ is said to be *convex* on S, if $S \subseteq \mathbb{R}^n$ is convex set, and for all $\mathbf{x}_1, \mathbf{x}_2 \in S$, $\lambda \in (0, 1)$ holds

$$f(\lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1 - \lambda)f(\mathbf{x}_2). \tag{3.5}$$

Following theorem describes preserving of convexity under composition.

THEOREM 3.8 (COMPOSITION OF CONVEX FUNCTIONS.).
Let $f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})$ be convex function on an set S. Then the following holds true:

(i) The composition $\sum_{j=1}^{k} \alpha_j f_j(\mathbf{x})$ is convex on S, if $\alpha_j > 0$ for $j = 1, \ldots, k$.

(ii) The function $f(\mathbf{x}) = \max\{f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})\}$ is convex on S.

Proof. Let S be a subset of $\mathbb{R}^n$, $\mathbf{x}_1, \mathbf{x}_2 \in S$, and $\lambda \in (0,1)$. Consider functions $f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})$ which are convex on S.

(i) Denote $f(\mathbf{x}) = \sum_{j=1}^{k} \alpha_j f_j(\mathbf{x})$. Then we may rewrite

$$\begin{aligned} f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) &= \sum_{j=1}^{k} \alpha_j f_j(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \\ \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) &= \lambda \sum_{j=1}^{k} \alpha_j f_j(\mathbf{x}_1) + (1-\lambda) \sum_{j=1}^{k} \alpha_j f_j(\mathbf{x}_2) \end{aligned}$$

Because of the convexity of functions f_j, from relation 3.5 we have

$$f_j(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f_j(\mathbf{x}_1) + (1-\lambda) f_j(\mathbf{x}_2) \qquad \forall j = 1, \ldots, k.$$

Clearly, if $\alpha_j > 0$ for each $j = 1, \ldots, k$, this relation remains the same after sum, i.e.

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2).$$

(ii) Denote $f(\mathbf{x}) = \max_{j \in J}\{f_j(\mathbf{x})\}$, where $J = \{1, \ldots, k\}$. Then

$$\begin{aligned} f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) &= \max_{j \in J}\{f_j(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2)\} \\ \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) &= \lambda \max_{j \in J}\{f_j(\mathbf{x}_1)\} + (1-\lambda) \max_{j \in J}\{f_j(\mathbf{x}_2)\} = \\ &= \max_{j \in J}\{\lambda f_j(\mathbf{x}_1) + (1-\lambda) f_j(\mathbf{x}_2)\} \end{aligned}$$

Since the inequality 3.5 is true for each function $f_j, j \in J$, the maximum preserves this property. Thus

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2).$$

□

The existence and uniqueness of an extreme (minimum or maximum) in nonlinear programming is guaranteed by Weierstrass theorem, and other fundamental theorems, which we give below. For proves and context within convex analysis see [3], [20], and [9].

THEOREM 3.9 (WEIERSTRASS).
Let $S \subseteq \mathbb{R}^n$ is a compact set (i.e. closed and bounded), and $f : S \to \mathbb{R}$ a continuous function on S. Then function $f(\mathbf{x})$ attains on S its minimum and maximum value.

Weierstrass theorem ensures, that under some conditions the minimum (or maximum) always exists. This classical result from real-valued analysis is always applicable to determine, if a objective function achieves an extreme value. More complicate is the task to find that value, or to find the optimal point $\mathbf{x}$. For a special cases of the objective function, this problem becomes easier.

THEOREM 3.10 (CONVEXITY AND GLOBAL OPTIMUM).
Let $S \subset \mathbb{R}^n$, $S \neq \emptyset$ be a convex set, and $f(\mathbf{x})$ be a convex function on S. If in $\bar{\mathbf{x}} \in S$ is achieved local minimum of function f, then in $\bar{\mathbf{x}}$ is also achieved global minimum of function f. Moreover, if $f(\bar{\mathbf{x}})$ is a strict local minimum, or f is strictly convex on S, then $f(\bar{\mathbf{x}})$ is a unique global minimum of f on S.

THEOREM 3.11.
Let $S \subset \mathbb{R}^n$, $S \neq \emptyset$ be a compact polyhedral set, and $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function on S. Then there exists $\bar{\mathbf{x}} \in S$ such that $f(\bar{\mathbf{x}})$ is a local minimum and $\bar{\mathbf{x}}$ is an extreme point of S.

THEOREM 3.12.
Let $S \subset \mathbb{R}^n$ be a nonempty convex set, and $f : \mathbb{R}^n \to \mathbb{R}$ be a convex and differentiable function. Then $\mathbf{x}^* \in S$ is a global minimum if and only if for all $\mathbf{x} \in S$ holds

$$\nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) \geq 0.$$

4. Network Flow

There are many various problems on graphs solvable with mathematical programming. One of the traditional, the *transportation problem*, deals with optimization of transportation in a network. A given amount of goods should be shipped from origins to destinations with respect to minimal transportation costs. Such problems are considered on bipartite graphs. A similar problem on an arbitrary digraph will be described in following section. In the section Minimal Cost Network Flow, we follow the book [2] inspired by the notation from [22], in section Multicommodity Minimal Cost Network Flow we cite the well known transportation model from [2], [9].

4.1. Minimal Cost Network Flow

Let us assume an arbitrary connected digraph $G = (N, A)$ with n nodes and m arcs. For every node $i \in N$, let us denote $A^{out}(i)$ the set of arcs directed out of a node i, and $A^{in}(i)$ the set of nodes directed into node i. The objective is to find an optimal way, how to transport available supply through the network with set of sources and targets.

With each node i in G we associate a number b_i. Thus, node i may represents the *source* ($b_i > 0$), the *target* ($b_i < 0$), or an *intermediate node* ($b_i = 0$). Let us denote x_a a flow through an arc $a \in A$, and c_a its cost per unit. We further assume that the total supply equals the total demand, i.e. $\sum_{i=1}^{n} b_i = 0$. If this is not the case, the problem can be always transformed to balanced one.

Definition 4.1 (MCF).
Consider all variables and coefficients as described above. The *Minimal Cost Network Flow problem* (MCF) is then defined as following linear program:

$$\begin{aligned} \min \quad & \sum_{a \in A} c_a x_a & & \\ \text{s.t.} \quad & \sum_{a \in A^{out}(i)} x_a - \sum_{a \in A^{in}(i)} x_a = b_i & \forall i \in N, & \\ & l_a \le x_a \le u_a & \forall a \in A. & \end{aligned} \tag{4.1}$$

The conditions $\sum_{a \in A^{out}(i)} x_a - \sum_{a \in A^{in}(i)} x_a = b_i$ are called *flow conservation.* Numbers l_a, u_a are lower and upper restrictions on flow in arcs, and are usually set to $l_a = 0$ and $u_a \ge 0$. Constraint of the type $0 \le x_a \le u_a$ is called *capacity constraint.*

If the transported supply are not arbitrarily divisible, we add the integrity constraint $\forall a \in A : \quad x_a \in \mathbb{Z}$.

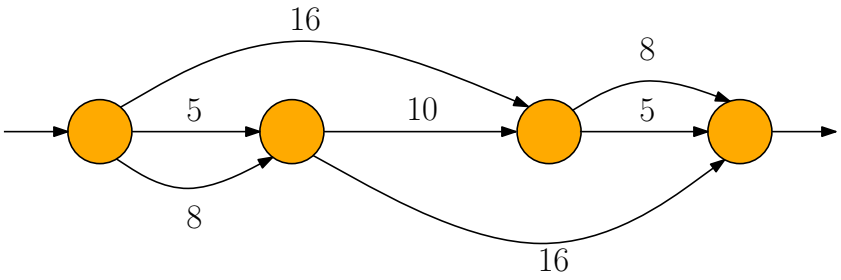
In the following, let us consider the problem mentioned above with the capacity constraints. Let us denote $\mathbf{x}$ the vector of flows, and let $\mathbf{c}, \mathbf{b}, \mathbf{u}$ be vectors of corresponding coefficients. The MCF can be rewritten using the node-arc incidence matrix $\mathbf{A}$ of the digraph[1]:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{o} \le \mathbf{x} \le \mathbf{u}. \end{aligned} \tag{4.2}$$

[1]Remember, that the columns in $\mathbf{A}$ and corresponding cells in all vectors are ordered in a same way to preserve the meaning of above described problem (4.1).

Thinking about the capacity constraints. It is useful to realize the following: Let us suppose, we should solve the MCF problem without any capacity constraints, with only one source $s \in N$ and one target $t \in N$. Obviously, the solution of this problem is given by solution of Shortest path problem from s to t (for the Shortest Path Problem see [2]). In opposite, solution of the problem (4.2), where for the upper bound $\mathbf{u}$ of the flow holds $\forall a \in A : \ u_a < \infty$, does not have to use the shortest path at all. This can be easily seen in the example below.

Example 4.1.
Let the capacity of each arc be $0 < u < \infty$, and we need to ship $2u$ units form source to target. As we see, the shortest path remains unused.



If half of the shipment uses the shortest path and the rest an other path, the cost is $44u$, although the minimal cost is $42u$

Network simplex method. Minimal Cost Network Flow (and other optimization problems on graphs) is just a special form of linear programming problem. Hereby, it is no surprise, that there exists a special form of simplex method for network problems, which uses specific features of these LPs. We just mention this method without concentrating us on it.

Nonlinearities in MCF. In a classical MCF pattern, the costs vector is constant, and does not depend on real flow in an arc. The flow is just bounded by the capacity constraint. This is not the right assumption in many branches of the humans actuation. For example, if the cost represents travelling time. It may take one car 20 minutes to go from one side of the town to the other, but the time surely increases if each driver decides to use the same way. For a hundred cars the cost may change to 40 minutes, etc.

Thus, the original MCF model must be redefined. A good way to describe such model similar to MCF is to express the traversing costs as a function of flow in an arc. In such model, the boundary is often not necessary, and the flow will be bounded by itself (or by the increasing costs). Outgoing from the original MCF, let us formulate this new nl-MCF model:

$$\begin{aligned} \min \quad & \mathbf{c}^T(\mathbf{x})\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{o}. \end{aligned} \tag{4.3}$$

Such model, where the costs explicitly depend on the flow, is a non-linear mathematical program.

4.2. Multicommodity Minimal Cost Network Flow

The Multicommodity Minimal Cost Network Flow, abbreviated *Multicommodity Flow* (MF), is optimization problem appearing in transportation branches. The MF problem can be viewed as generalization of the Minimal Cost Network Flow. Hence, the object of MF is to minimize the cost of the flow for multiple commodities going from a specific source to a specific target.

In a graph G, we should transport different commodities, whose set will be denoted $\mathcal{C}$. For each commodity $i \in \mathcal{C}$, denote $\mathbf{x}_i$ the flow vector, $\mathbf{c}_i$ the cost of flow, $\mathbf{b}_i$ the supply vector, and $\mathbf{u}_i$ the upper bound on flow of i in arcs of the graph. The graph is given by its node-arc incidence matrix $\mathbf{A}$, and every commodity can use all arcs (with respect to its capacity $\mathbf{u}_i$). Moreover, assume that each arc has its maximal capacity on flow. Thus, denote $\mathbf{u}$ the upper bound on the sum of flow of all commodities.

Definition 4.2.
With respect to assumptions we made in this section so far, *Multicommodity Flow* will denote the following linear program:

$$\begin{aligned} \min \quad & \sum_{i \in \mathcal{C}} \mathbf{c}_i^T \mathbf{x}_i \\ \text{s.t.} \quad & \sum_{i \in \mathcal{C}} \mathbf{x}_i \leq \mathbf{u}, \\ & \mathbf{A}\mathbf{x}_i = \mathbf{b}_i \qquad \forall i \in \mathcal{C}, \\ & \mathbf{o} \leq \mathbf{x}_i \leq \mathbf{u}_i \quad \forall i \in \mathcal{C}. \end{aligned} \tag{4.4}$$

Nonlinearities in MF. As in the MCF model, cost function in MF can be for special cases transformed to a non-linear one, explicitly depending on the total flow $\mathbf{x}$ in arcs. Such models are considered in section 4.3 below.

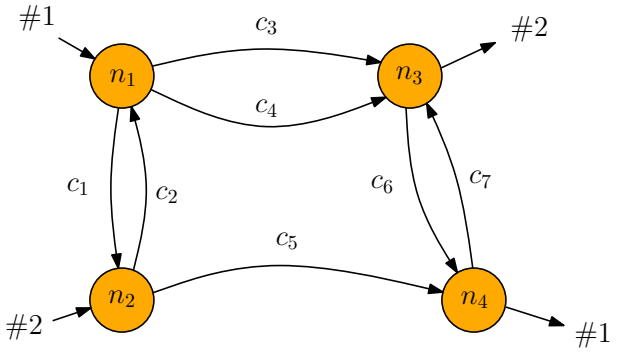
4.3. Traffic Assignment Problem

The *Traffic Assignment Problem* (TAP) is a network flow problem focused on finding an optimal distribution of flow in a given network. A typical object of study are traffic networks where the minimal cost flow is affected by interests of all user of the network. Here, a particular user is looking for minimization of his costs, and the goal of TAP is to determine the effectiveness of this behaviour.

Example 4.2 (One simple model).
Let us consider a small network consisting of only four nodes and seven arcs. Assume, that drivers are divided into two groups depending on their starting and terminating node.

#	direction	number of drivers
1	(n_1, n_4)	8
2	(n_2, n_3)	4

Denote set of this groups by $\mathcal{C}$. Let x_{ai} be the flow through arc a of drivers belonging to $i \in \mathcal{C}$. Because the capacity of all roads is not quite sufficient (not every road is a highway), the time c_a of going the road a fully depends on its utilization, on the total flow $f_a = \sum_{i \in \mathcal{C}} x_{ai}$. The sum over all arcs represents the total time spend in the system. Thus $z = \sum_a c_a(f_a) f_a$ is the objective function to be minimized.



ONE SIMPLE MODEL

Note, that for travel times

$$c_1 = 8 + 3f_1, \quad c_2 = 8 + 4x_2, \quad c_3 = 10 + 2x_3, \quad c_4 = 2 + 5x_4,$$
$$c_5 = 6 + 4f_5, \quad c_6 = 2 + 5x_6, \quad c_7 = 3 + 4x_7,$$

the flow $\mathbf{f}$ in arcs is $(3, 2, 4, 3, 5, 5, 2)$, which is feasible, and the total travel time is $z = 493$.

Moreover, it is easy to see, that no one of the vehicles can decrease its travel time by choosing the other road. (For example, costs of all paths for the 1^{st} group are $43, 44, 45$ minutes.) Although no one driver can improve his state, this flow is not the "minimal cost flow" since the flow $\tilde{\mathbf{f}} = (4, 3, 4, 3, 5, 4, 1)$ achieves the total cost $z = 488$. □

System optimum. As we see, in traffic models we must distinguish two approaches, which have in general different optimal solutions. With *system optimum* (SO) we mean finding the solution that minimizes the total flow-cost in the network. This concept corresponds to MCF or MF described in sections above. Instances of such class of problems are networks where the flow is controlled by some central authority, for example in railway networks, military supplies, or flight control.

User equilibrium. An other approach to the flow problematic is given by using principles of the game theory. We suppose, that all users in the network want to minimize their own costs, subject to every other user doing the same. (Under an user, so far we understand an "infinitesimal user", so that flow through arcs can take any real value.) Obviously, users try to find less utilized paths, and thus they equilibrate the costs. If no user can decrease his cost by choosing another path, we call this state *user equilibrium* (UE), or also *Wardrop equilibrium*. UE is the main objective of TAP, capturing more realistic behaviour of users.

Wardrop [21] formulated in 1952 two alternative criteria of the optimal traffic flow. One of these approaches described above rises up from the game theory aspects. (For a short comparison between user equilibrium and Nash equilibrium in game theory see appendix A.) We cite [21, page 344.]:

FIRST WARDROP'S PRINCIPLE - UE.
The journey times on all the routes actually used are equal, and less than those which would be experienced by a single vehicle on any unused route.

SECOND WARDROP'S PRINCIPLE - SO.
At equilibrium the average journey time is a minimum.

The first one corresponds to user equilibrium. If UE is achieved, no user can improve his cost by choosing other path. The second principle refers to system optimum, minimization of the total travel time, and thus to the best utilization of the system.

4.4. Static Traffic Assignment Problem

The goal of a TAP is to allocate all users with fixed origin-destination pairs to arcs in a graph, in order to attain SO or UE. In the case of static TAP, all parameters of the network remain without any change. Thus in this sense, in static TAP no randomness appears, and the traffic flow is usually restricted to a specific time period (e.g. in the rush hour) due to omit changes in the flow.

We consider a network flow problem on a graph $G = (N, A)$. Each user of the network can be represented as a pair of origin and destination (s_i, t_i), which we understand as particular commodity. As in the previous section, we denote the set of commodities by $\mathcal{C}$. Further, denote $\mathcal{P}_i$ the set of all simple $s_i - t_i$ paths (see definition 2.12). We always assume $\forall i \in \mathcal{C}\ \mathcal{P}_i \neq \emptyset$.

For a given commodity $i \in \mathcal{C}$, we denote by $\mathbf{x}_i = (x_{i1}, \ldots, x_{in})$ the flow-vector of i. The state of the network is characterized by the vector of *(total) flow* $\mathbf{f} = (f_a)_{a \in A}$, where $f_a = \sum_{i \in \mathcal{C}} x_{ia}$ is the total flow of all commodities in arc a. Suppose, cost of traversing an arc $a \in A$ is given by the *cost function* $c_a(f_a)$. As mentioned in example above, the *cost of a flow* $C(\mathbf{f})$ is achieved by the function of following form

$$C(\mathbf{f}) = \sum_{a \in A} c_a(f_a) f_a.$$

This is the price for units travelling through arcs, with cost dependent on the flow. Minimization of this function leads to SO, exactly according to the Second Wardrop's principle. Ahead to define the UE, for a specific flow $\mathbf{f}$ term the restriction of the cost $C(\mathbf{f})$ to the path P

$$C_P(\mathbf{f}) = \sum_{a \in A, a \in P} c_a(f_a) f_a.$$

Definition 4.3 ([19]).
Let $\mathbf{f}$ be a feasible flow of a network flow problem. Flow $\mathbf{f}$ is a *user equilibrium* (UE) if, for every commodity $i \in \mathcal{C}$ and every pair of paths $P, P' \in \mathcal{P}_i$, the following holds

$$C_P(\mathbf{f}) \leq C_{P'}(\mathbf{f}).$$

In other words, all paths of given commodity used by a user equilibrium have equal cost.

Theorem 4.1.
Assume a network flow problem on a graph G with cost functions $c_a(f_a)$. Then:

(i) There exists at least one user equilibrium.

(ii) If $\mathbf{f}$ and $\mathbf{f}'$ are user equilibria, then $c_a(f_a) = c_a(f'_a)$ for every arc a.

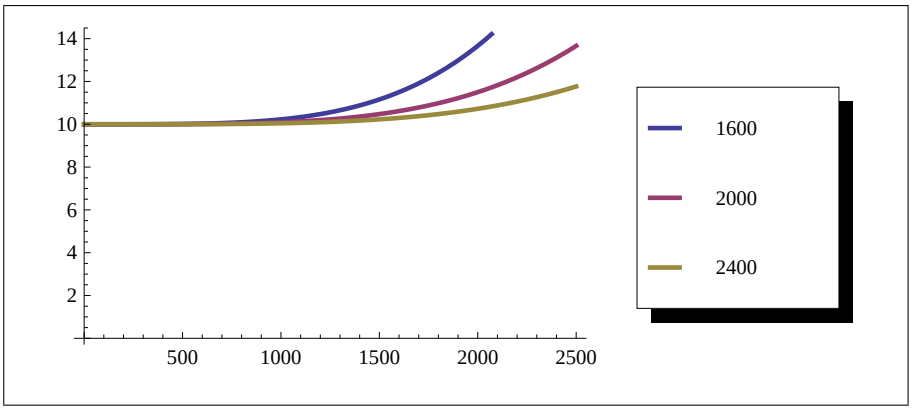
Prove of foregoing theorem can be found in Beckmann, McGuire, and Winsten [4]. They also showed, using the Karush-Kuhn-Tucker conditions (see [3]), that the UE are precisely the flows that minimize the function $\bar{z} = \bar{z}(\mathbf{f})$

$$\bar{z} = \sum_{a \in A} \int_0^{f_a} c_a(s)\,\mathrm{d}s. \tag{4.5}$$

Beckmann's model. Commonly used is so-called Beckmann's model [4], where the travel time for $a \in A$ only depends on the flow. Beckmann, McGuire and Winsten formulated a TAP, where the cost functions $c_a(f_a)$ (called *latency functions*[2]) are arbitrary functions, which are convex, continuous, nonnegative, and nondecreasing. Capacity constraints are not taken into account, and the violation of capacity is penalized by the cost function itself. Note, that this allows solutions, where capacity constraints are exceeded. Other variant of TAP with cost function developed by Nesterov & de Palma can be found on the page 39.

Typical latency functions are the BPR-functions (developed in 1964 by Bureau of Public Road) with two parameters $\alpha, \beta \geq 0$. To each arc is assigned its capacity u_a (vehicles per hour) and a nonnegative number called *free travel time*, denoted t_a^0. For simulating the real road congestions, parameters α, β are usually set to 0.15 and 4.

$$c_a(f_a) = t_a^0 \left(1 + \alpha \left(\frac{f_a}{u_a}\right)^{\beta}\right).$$



BPR LATENCY FUNCTIONS FOR DIFFERENT CAPACITIES

According to MCF and MF network problems, for a given commodity $i \in \mathcal{C}$, $\mathbf{b}_i = (b_{i1}, b_{i2}, \ldots, b_{im})$ corresponds to the number of users travelling during a given period through the network (4.2, 4.4). The vector $\mathbf{b}_i$ must be balanced, i.e. $\sum_{j \in N} b_{ij} = 0$. The topology of the graph is expressed by node-arc incidence matrix $\mathbf{A}$.

[2]Although the latency function is in literature often denoted by $l = l(f_a)$, we continue with our notation to keep the continuity between preceding chapters.

Definition 4.4 (B-SO).
With respect to the First Wardrop's principle, a SO in the Beckmann's model is the solution of the following optimization problem

$$\begin{array}{rll} \min_{\mathbf{f}} & \sum\limits_{a\in A} c_a(f_a)f_a & \\ \text{s.t.} & f_a = \sum\limits_{i\in\mathcal{C}} x_{ia} & \forall a \in A, \\ & \mathbf{A}\mathbf{x}_i = \mathbf{b}_i & \forall i \in \mathcal{C}, \\ & \mathbf{x}_i \geq \mathbf{o} & \forall i \in \mathcal{C}. \end{array} \tag{4.6}$$

According to result of Beckmann et al. from page 36 (equation 4.5), as also to [4, chapter 4.] and [6], the "oposite" of (B-SO) may be stated in a following way.

Lemma 4.1 (B-UE).
Under Beckmann's assumptions, user equilibrium is solution of the following optimization problem

$$\begin{array}{rll} \min_{\mathbf{f}} & \sum\limits_{a\in A} \int\limits_0^{f_a} c_a(s)\,\mathrm{d}s & \\ \text{s.t.} & f_a = \sum\limits_{i\in\mathcal{C}} x_{ia} & \forall a \in A, \\ & \mathbf{A}\mathbf{x}_i = \mathbf{b}_i & \forall i \in \mathcal{C}, \\ & \mathbf{x}_i \geq \mathbf{o} & \forall i \in \mathcal{C}. \end{array} \tag{4.7}$$

For the proof see [4]. UE and SO as solutions of (4.6, 4.7) always exist as long as the set $\{(\mathbf{x}_i)_{i\in\mathcal{C}} | \mathbf{A}\mathbf{x}_i = \mathbf{b}_i, \mathbf{x}_i \geq \mathbf{o} \quad \forall i \in \mathcal{C}\}$ is nonempty (see theorem 4.1).

Notation. Further, we denote the objective function of (B-SO) by $z = z(\mathbf{f})$. Obviously, the cost of the flow equals to value of this objective function, i.e. $C(\mathbf{f}) = z(\mathbf{f})$, while for objective function $\bar{z}$ of (B-UE) this equality does not have to come true.

Beckmann's model viewed as MF. Obviously, the Beckmann's model corresponds to Multicommodity flow problem with nonlinear objective function without any capacity constraints (for one commodity is related to Minimal Cost Network Flow). Variable f_a is redundant, just used like abbreviation for the first constraint. The two latter constraints are taken from Multicommodity Flow, expressing the properties of the flow. For the comparison, see below

(MF)

$$\begin{array}{rll} \min & \sum\limits_{i\in\mathcal{C}} \mathbf{c}_i^T\mathbf{x}_i & \\ \text{s.t.} & \sum\limits_{i\in\mathcal{C}} \mathbf{x}_i \leq \mathbf{u}, & \\ & \mathbf{A}\mathbf{x}_i = \mathbf{b}_i & \forall i \in \mathcal{C}, \\ & \mathbf{o} \leq \mathbf{x}_i \leq \mathbf{u}_i & \forall i \in \mathcal{C}. \end{array}$$

(B-SO)

$$\begin{array}{rll} \min_{\mathbf{f}} & \sum\limits_{a\in A} c_a(f_a)f_a & \\ \text{s.t.} & f_a = \sum\limits_{i\in\mathcal{C}} x_{ia} & \forall a \in A, \\ & \mathbf{A}\mathbf{x}_i = \mathbf{b}_i & \forall i \in \mathcal{C}, \\ & \mathbf{x}_i \geq \mathbf{o} & \forall i \in \mathcal{C}. \end{array}$$

The objective function for MF can be in nonvector form written as

$$\sum_{i\in\mathcal{C}} \mathbf{c}_i^T\mathbf{x}_i = \sum_{i\in\mathcal{C}}\sum_{a\in A} c_{ia}x_{ia}.$$

In MF, we usually assume the costs are the same for all commodities, and thus $c_{ia} = c_a \ \forall i \in \mathcal{C}$. If we appoint the substitution for total flow in an arc f_a, for the Beckmann's model we get following objective function

$$\sum_{a\in A} c_a(\sum_{i\in\mathcal{C}} x_{ia}) \sum_{i\in\mathcal{C}} x_{ia} = \sum_{a\in A}\sum_{i\in\mathcal{C}} c_a(\sum_{i\in\mathcal{C}} x_{ia})x_{ia}.$$

From the theorem 3.5 follows, that matrix $\mathbf{A}$ is totally unimodular matrix, and the theorem 3.6 implies that the polytope defined by the last two constraints is integral for every commodity $i \in \mathcal{C}$. This is exactly the same as in MF, except to nonlinear cost functions, that lead in generality to nonintegral solution.

Price of anarchy. To measure differences between UE and SO, in order to determine the usefulness of the network, the *price of anarchy* is defined.

Definition 4.5 (Price of anarchy).
Suppose a network consisting of graph $G = (N, A)$, cost functions $(c_a)_{a\in A}$, and demand vectors $\mathbf{b}_i$ for all occurring commodities in the set $\mathcal{C}$. Let $\mathbf{f}^*$ be the solution of (B-SO), and $\overline{\mathbf{f}}^*$ the solution of (B-UE). We define the *price of anarchy* $\rho = \rho(G, (\mathbf{b}_i)_{i\in\mathcal{C}}, (c_a)_{a\in A})$ as the following ratio

$$\rho = \frac{C(\overline{\mathbf{f}}^*)}{C(\mathbf{f}^*)}.$$

Price of anarchy measures "how far" is an UE from the best utilization of the system. Since $\mathbf{f}^*_{SO}$ is the best usage of the system, we can easily show that $\rho \geq 1$ always holds.

Let $\mathcal{F}$ denote a nonempty set of cost functions. Roughgarden in his work [19] showed, that if $\mathcal{F}$ is a set of polynomial cost functions with nonnegative coefficients and degree at most p, the price of anarchy ρ is bounded from above by *Pigou bound* $\alpha(\mathcal{F})$

$$\alpha(\mathcal{F}) = \left(1 - p(p+1)^{-(p+1)/p}\right)^{-1}.$$

Thus, for linear cost functions, the upper bound is $4/3$, and for BPR-functions with $\beta = 4$, the bound is approximately 2.15.

Criticism to Beckmann's model. Some authors highlight the imperfection of the model, referring to unconstrained capacity violation. Since the static TAP supposes model of the traffic flow in a time period, the overflow of capacity implies in reality, that some part of the flow will have to move to following period. Because of this, the initiative rose up, to develop more adaptable model. This led for example to Nesterov and de Palma's model decsribed in below.

One way how to avoid some problem with capacity violation is an extension of the original Beckmann's model. We add a bunch of new constraints

$$g_j(\mathbf{f}) \leq 0 \qquad \forall j \in J, \tag{4.8}$$

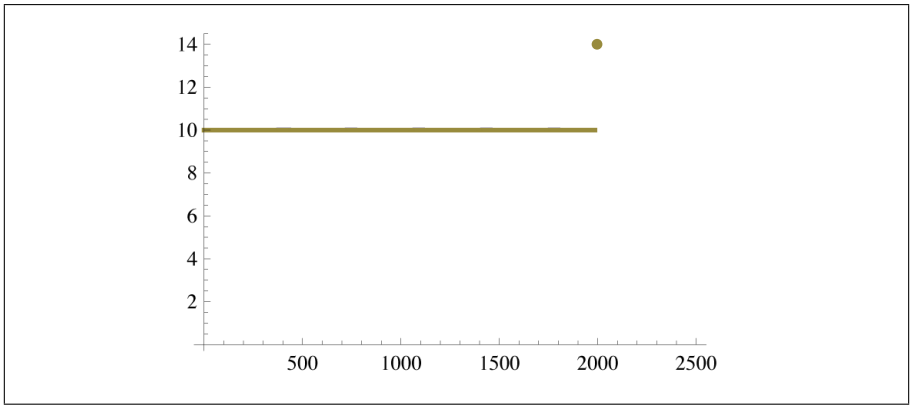
where the function g_j is convex and continuous differential function, and the set J can be a subset of sets $A, \mathcal{C}$, or N. Hereby, we may consider a special case of the capacity constraint $g_a(\mathbf{f}) = f_a - u_a$. Further, we denote a mathematical program (B-SO), (B-UE) with these additional constraints as (B-SOext), (B-UEext), respectively.

Another Variants of TAP Model

Nesterov & de Palma's model. Sometimes, the commonly used Beckmann's model is in TAP models replaced by the model formulated by Nesterov and de Palma. Here, the capacity u_a of an arc cannot be exceeded, and the tightness to the capacity is penalised only if this is achieved. Mathematically described, Nesterov and de Palma developed cost functions $\mathbf{t}(\mathbf{f}) = (t_a)_{a\in A}$ with following property:

$$t_a = \begin{cases} t_a = t_a^0 & \text{if } f_a < u_a, \\ t_a \geq t_a^0 & \text{if } f_a = u_a. \end{cases}$$

The total travel time is defined as $\sum_{a\in A} f_a t_a = \mathbf{t}^T\mathbf{f}$.



Nesterov and de Palma's cost function

Nesterov and de Palma discussed some weaknesses of Beckmann's model with respect to TAP fundamentals. Since every TAP is usually restricted to a state of network in a specific time period, if there is a congestion of a road, users may leave the road in a following period. This fact is by Beckmann's model not token into account. On the other hand, Nesterov and de Palma use function, which can be discontinuous at the point of capacity.

In our text, we do not consider TAP according to that one defined by Nesterov & de Palma, but still we find it useful to mention it as another variant to Beckmann's model. A particular comparison of Nesterov & de Palma's and Beckmann's model was done in [6].

4.5. Example of Static TAP in GAMS

To show, that results in Example 4.2 correspond to (B-SO) and (B-UE) stated above, we demonstrate the two above declared mathematical programs in software GAMS (for short introduction to GAMS see appendix). Further, we consider in our text Beckmann's models of TAP (most often with BPR-functions). Moreover, the costs in Example 4.2 respond to special case of BRP-function, to linear functions with unit arc capacity.

In software GAMS the problem is described as follows:

```
sets     I set of arcs /arc1*arc7/
         N set of nodes /node1*node4/
```

```
         J set of commodities /com1,com2/;

table Demand(N,J) demand at node n of commodity j
                 com1       com2
         node1    8
         node2               4
         node3              -4
         node4   -8                ;

table A(N,I) incidence matrix
                 arc1    arc2    arc3    arc4    arc5    arc6    arc7
         node1    1       -1       1       1
         node2   -1        1                       1
         node3                    -1      -1              1      -1
         node4                                   -1      -1       1  ;

parameter FreeTime(I) free travel time of arc i
         /arc1 8, arc2 8, arc3 10, arc4 2, arc5 6, arc6 2, arc7 3/;
parameter Beta(I) multiplication coeficient of arc i
         /arc1 3, arc2 4, arc3 2, arc4 5, arc5 4, arc6 5, arc7 4/;
parameters zzUE    cost of UE flow
           poa     price of anarchy
           xprint(I,J) x variable for output;

variables        x(I,J)  flow through arc i of com. j
                 zSO     SO-total time cost
                 zUE     UE-total time cost
                 fSO(I)  OS flow through arc i
                 fUE(I)  UE flow through arc i;
integer variable x;

equations        costSO        SO-objective function
                 costUE        UE-objective function
                 flowCon(N,J)  flow conservation constraints
                 flowSO(I)     OS flow definition
                 flowUE(I)     UE flow definition;
flowSO(I)..      fSO(I) =e= sum(J, x(I,J));
flowUE(I)..      fUE(I) =e= sum(J, x(I,J));
costSO..         zSO =e= sum(I,(FreeTime(I)+Beta(I)*fSO(I))*fSO(I));
costUE..         zUE =e= sum(I,(FreeTime(I)+Beta(I)*fUE(I)/2)*fUE(I));
flowCon(N,J)..   sum(I,A(N,I)*x(I,J)) =e= Demand(N,J);

model tapOS /costSO, flowSO, flowCon/;
model tapUE /costUE, flowUE, flowCon/;
solve tapOS using minlp minimizing zSO ;
display x.l,zSO.l;
xprint(I,J)=x.l(I,J);

solve tapUE using minlp minimizing zUE ;
zzUE=sum(I,(FreeTime(I)+Beta(I)*fUE.l(I))*fUE.l(I));
```

```
display x.l,zUE.l,zzUE;

poa=zzUE/zSO.l;
display poa;
display fSO.l,fUE.l;
```

This model is attached to this text on the CD under name *myTAP1.gms*. Using solver for mixed integer nonlinear programming (see BARON, DICOPT, and ALPHAECP in appendix B), we get exactly the result as presented before:

```
----------------------Traffic Assignment Problem: myTAP1----------------------

The Flow
  -----------system optimum----------- ------------user equilibrium-----------
           com1    com2   total flow  |          com1    com2   total flow
                                      |
  arc1     4.00    0.00          4.00 |arc1      3.00    0.00          3.00
  arc2     0.00    3.00          3.00 |arc2      0.00    2.00          2.00
  arc3     4.00    0.00          4.00 |arc3      2.00    2.00          4.00
  arc4     0.00    3.00          3.00 |arc4      3.00    0.00          3.00
  arc5     4.00    1.00          5.00 |arc5      3.00    2.00          5.00
  arc6     4.00    0.00          4.00 |arc6      5.00    0.00          5.00
  arc7     0.00    1.00          1.00 |arc7      0.00    2.00          2.00

Cost of the Flow:
           cost zSO =    488.00                 cost zzUE =    493.00
                                                (cost zUE =    312.50)

Price of Anarchy:
           poa =   1.01025
```

Whenever the cost of OS solution is $z^*_{OS} = 488$, the cost of user equilibrium is $z^*_{UE} = 493$, and thus the price of anarchy $\rho \approx 1.01$.

5. Improved TAP

5.1. Stochastic Programming

So far, we have discussed a deterministic mathematical programs and deterministic models. But in a more realistic models, some parameters of the network problems can be "unknown", or the users of the network may have incomplete information about the network in the time they should decide. This variability of the problem can be modelled as mathematical program in the presence of a randomness. We call this branch *Stochastic (mathematical) programming*, and a corresponding model a *stochastic program*, which we here define:

Definition 5.1 (Stochastic program).
Let $\Omega, \mathcal{S}, P$ be a *probability space*, and $\boldsymbol{\xi} : \Omega \to \mathbb{R}^K$ a *random vector*. The following problem is said to be a *stochastic (mathematical) program* (SP)

$$\min_{\mathbf{x}} f(\mathbf{x}, \boldsymbol{\xi}) \text{ s.t. } \mathbf{x} \in X(\boldsymbol{\xi}). \tag{5.1}$$

Compare this definition with deterministic MP, page 21. Since the interpretation of SP in not clear before knowing the realisation of $\boldsymbol{\xi}$, the problem 5.1 is not well defined. Usually, the stochastic programs are reformulated into a *deterministic equivalents* that correctly interpret random variables. We further assume, that the objective function is measurable, and the expectation $Ef(\mathbf{x}, \boldsymbol{\xi})$ exists for all $\mathbf{x}$. As $\boldsymbol{\xi}$ is an $\mathcal{S}$-measurable mapping, it induces a probability distribution on $\mathbb{R}^K$. In a SP we assume, that the distribution of $\boldsymbol{\xi}$ is known. For further purposes, denote support of $\boldsymbol{\xi}$ as $\Xi \subset \Omega$.

Next, we introduce two main principles, how to deal with stochastic programs. Further we use the following notation: if the realisation of the random vector $\boldsymbol{\xi}$ is known, we denote this value $\boldsymbol{\xi}^s$. The two principles of solving stochastic programs are based on different approaches:

- either the realisation $\boldsymbol{\xi}^s$ precedes the decision of $\mathbf{x}$,
- or the decision of $\mathbf{x}$ precedes the realisation $\boldsymbol{\xi}^s$.

Remark. Recall, we distinguish two types of random variables: *discrete* and *continuous*. For the first case, if the number of realisations is finite ($|\Xi| < \infty$), we denote these by S and talk about a *scenarios* of $\boldsymbol{\xi}$.

Wait-and-see Approach (WS)

When a decision $\mathbf{x}$ is made after observing the realisation of the random vector $\boldsymbol{\xi}$, we call this approach *wait-and-see*. WS models are solved with respect to variable $\mathbf{x}$, considering a random vector as a parameter. The solution $\mathbf{x}$ and the objective function z are functions of $\boldsymbol{\xi}$, $\mathbf{x} = \mathbf{x}(\boldsymbol{\xi})$ and $z = z(\boldsymbol{\xi})$, and thus a random vectors (variables). An interesting question to ask is about the distributions of $\mathbf{x}(\boldsymbol{\xi})$ and $z(\boldsymbol{\xi})$. This has to be determined, because since now we deal with variables as with random variables, etc.

WS deterministic reformulation. Let the SP be given as 5.1. We define its *wait-and-see (WS) deterministic reformulation* as

$$\min_{\mathbf{x}(\boldsymbol{\xi})} f(\mathbf{x}(\boldsymbol{\xi}), \boldsymbol{\xi}) \text{ s.t. } \mathbf{x}(\boldsymbol{\xi}) \in X(\boldsymbol{\xi}). \tag{5.2}$$

We denote the minimal objective function value as z^*_{WS} and minimum as $\mathbf{x}^*_{WS}$. Note, that still we have $z^*_{WS} = z^*_{WS}(\boldsymbol{\xi})$ and $\mathbf{x}^*_{WS}(\boldsymbol{\xi})$. Thus, for every realisation $\boldsymbol{\xi}$ we get solution of 5.2, and we can deal with the set of all solution for every realisation $\boldsymbol{\xi}$ as with statistical data set.

Here-and-now Approach (HN)

If we must make the decision before the observation of $\boldsymbol{\xi}$, we call this case *here-and-now* (HN) approach. The decision $\mathbf{x}$ is then the same for all possible future realisations of the randomness. SP primarily deals with HN approach, because the typical decision situation is described by the lack of observations.

EV deterministic reformulation. Let the SP be given as 5.1. We define its here-and-now *expected value (EV) deterministic reformulation* as

$$\min_{\mathbf{x}} f(\mathbf{x}, E\boldsymbol{\xi}) \text{ s.t. } \mathbf{x} \in X(E\boldsymbol{\xi}), \tag{5.3}$$

where $E\boldsymbol{\xi}$ is the expected value of $\boldsymbol{\xi}$. We denote the minimal objective function value as z^*_{EV} and minimum as $\mathbf{x}^*_{EV}$.

Remark. Note, that for EV we can solve MP with replacing random vector with values, that may appear with zero probability. (Especially with expected value of discrete random vector.) Although this fact, EV model is reasonable.

EO deterministic reformulation. Let the SP be given as 5.1. We define its here-and-now *expected objective (EO) deterministic reformulation* as

$$\min_{\mathbf{x}} E\left[f(\mathbf{x}, \boldsymbol{\xi})\right] \text{ s.t. } \mathbf{x} \in X, \tag{5.4}$$

We denote the minimal objective function value by z^*_{EO} and the minimum as $\mathbf{x}^*_{EO}$.

Remark. According to results reached in stochastic programming, we must state, that with larger variance of random vector, the solution z^*_{EV} becomes more optimistic than z^*_{EO}.

The outcome of EV and EO are solutions $\mathbf{x}^*$ which are used for every realisation of the randomness. Thus, for "fixed"solution $\mathbf{x}^*$ and all random vectors $\boldsymbol{\xi}$ we get set of result, which we can eventually compare with WS approach.

Remark. Note, that due to implementation of all following stochastic problems in GAMS software, the number of scenarios is always finite. Hereby, we replace the expected values $E\boldsymbol{\xi}$ by weighted sum

$$\sum_{s \in S} p_s \boldsymbol{\xi}^s$$

where p_s is the probability of the realisation $\boldsymbol{\xi}^s$. This sum simplifies for uniform distribution into $\frac{1}{|S|}\sum_{s\in S}\boldsymbol{\xi}^s$. Although we further write $E\boldsymbol{\xi}$ in general, we implement it as the weighted sum.

5.2. Stochastic TAP

In this section, we consider, that not all parameters are known in our TAP model, and we discuss the problems form different approaches. The randomness may appear as in the cost function $c_a(f_a, \boldsymbol{\xi})$ as on the other places, i.e. in constraint, etc.

Randomness in the Objective Function

First we suppose, that the random vector can appear in our cost functions, i.e. each arc is affected by some unknown phenomenon ($z = z(\mathbf{f}, \boldsymbol{\xi})$ and $\bar{z} = \bar{z}(\mathbf{f}, \boldsymbol{\xi})$). Imagine, although each user driving through the network has an idea about the distance and congestion on each road, one can never predict concrete situation. There may happen an accident slowing the traffic, or on the other hand the flow of user can be must faster than predicted.

Assume, each cost function c_a depends now on random vector $\boldsymbol{\xi}$, i.e. $c_a = c_a(f_a, \xi_a)$. Looking back to Beckmann's model, now we suppose the cost functions as BPR-functions in the form

$$c_a(f_a, \xi_a) = k_a + q_a(f_a)^p + \xi_a, \qquad p \geq 0.$$

For all mathematical programs solving the user-equilibrium-related flow, we use a simple reformulation of the objective function, consequently the cost functions. Since they are easy integrable, we lie the objective function $\bar{z}$ in a following form

$$\bar{z} = \sum_{a\in A}\int_0^{f_a} c_a(s, \xi_a)\ \mathrm{d}s = \sum_{a\in A}(k_a + \frac{q_a}{p+1}(f_a)^p + \xi_a)f_a$$

Since, we have no additional information about the distribution of $\boldsymbol{\xi}$, we will usually consider uniform distribution with expected value $E\boldsymbol{\xi}$ equal to zero.

WS. For all $\boldsymbol{\xi}^s \in \Xi$ we get a deterministic reformulation of (B-SO) and (B-UE)

$$\begin{aligned} z_{WS}(\mathbf{f}, \boldsymbol{\xi}^s) &= \sum_{a\in A}(k_a + q_a(f_a)^p + \xi_a^s)f_a, \\ \bar{z}_{WS}(\mathbf{f}, \boldsymbol{\xi}^s) &= \sum_{a\in A}\int_0^{f_a}(k_a + q_a t^p + \xi_a^s)\ \mathrm{d}t. \end{aligned}$$

EV and EO. Note, that function $c_a(f_a, \xi_a)$ is linear in variable ξ_a. Hence, as EV so EO deterministic reformulations result in the same MP for both problems (B-SO) and (B-UE). See the short derivation of objective functions $z, \bar{z}$ modifications

$$\begin{aligned} z_{EO}(\mathbf{f}) &= E\sum_{a\in A}(k_a + q_a(f_a)^p + \xi_a)f_a = \sum_{a\in A}[(k_a + q_a(f_a)^p)f_a + f_a E\xi_a] = z_{EV}(\mathbf{f}) \\ \bar{z}_{EO}(\mathbf{f}) &= E\sum_{a\in A}\int_0^{f_a}(k_a + q_a t^p + \xi_a)\ \mathrm{d}t = \sum_{a\in A}\int_0^{f_a}(k_a + q_a t^p + E\xi_a)\ \mathrm{d}t = \bar{z}_{EV}(\mathbf{f}). \end{aligned}$$

Example 5.1 (Comparison of WS and HN results).
We used software GAMS as a computing tool to compare the prices of anarchy of WS and HN approaches. We based the following model on the preceding "One Simple Model" from page 39, considering a random vector uniformly distributed between values -2 and 2, i.e. $\boldsymbol{\xi} \sim U(-2, 2)$. For a 200 realisations of random vector $\boldsymbol{\xi}$ we counted the price of anarchy for HN model, and compared it with the expected value of prices of anarchy reaching from WS models. Folder containing this model is included under the name *myTAP2*. We present just the main part of our results, which are attached in the file *OUT.txt*.

We generated $|S| = 200$ realisations of random vector, i.e. table over $\mathbb{R}^{|S|\times|A|}$, where $|A| = 7$ is number of arc in the network. The vector of expected values is as follows

```
Expected Value Vector: Exi(I) = (-0.03,-0.07,-0.06,0.054,0.000,0.051,-0.20)
```

In the case of here-and-now approach (as we derived earlier) the randomness should affect both SPs only through expected value in the objective function. For random vector with $E\boldsymbol{\xi} = 0$, the price of anarchy ρ_{HN} converges to original value $\rho \approx 1.0102$ for a big number of realisations. Comparing prices ρ_{HN} for all realisations of vector $\boldsymbol{\xi}$ with fixed flow-results from HN approach, we got a data set to deal with. This data show, that there is a relative small variance.

```
--------------------------
here-and-now
-------------
  Cost of the Flow:          zSO_HN  =    487.574
                             zzUE_HN =    505.644

  HN Price of Anarchy:       PoA     =    1.03706

  Variance of PoA for all realisations of xi:
                         var_xi(PoA) =    0.00004
```

Although ρ_{HN} draws near to ρ for big $|S|$, the price of anarchy ρ_{WS} given by WS approach is already much better estimation of ρ for a small number of realisations. While having higher variance, the mean ρ_{WS} is much better estimation of original price of anarchy ρ. See below and compare

```
--------------------------
wait-and-see
-------------
  Expected value of OS solutions:          487.79
  Expected value of UE solutions:          494.54
  Expected value of PoA:                  1.01381
  Variance of PoA:                        0.00014
```

Randomness in the Constraints

Suppose, for example, that a given network is just a part of a huge network. At nodes, where our network is connected to the rest part, it may occur a random number of arrival for those users, that want to continue in their way (like a real life situation from a small town in a densely populated area). We state the problem (see models 4.6, 4.7) with the following type of constraints

$$\mathbf{A}\mathbf{x}_i = \mathbf{b}_i(\boldsymbol{\xi}) \quad \forall i \in \mathcal{C},$$

where always for set of nodes N and for all commodities $\sum_{j\in N} b_{ij}(\boldsymbol{\xi}^s) = 0$ (thus the flow conservation is ensured). This can be interpreted as uncertainty in the quantity of drivers representing each commodity. Due to the meaningfulness, we will consider the right-hand side vector $\mathbf{b}(\boldsymbol{\xi})$ in following special form

$$b_{ij}(\xi_j) = \begin{cases} b_{ij} \pm \xi_j & \text{if } b_{ij} > 0 \text{ or } b_{ij} < 0, \\ 0 & \text{else.} \end{cases} \tag{5.5}$$

So the intermediate nodes remain preserved as same as all commodity-flows directions. According to sense of our model, we naturally assume $\boldsymbol{\xi}$ having discrete uniform distribution. Since each commodity is represented by its source s_i and target t_i, the flow conservation constraint forces the volume of user to be equal at the beginning and at the end. Thus, if randomness results in decrease of drivers coming in, there must be the same declination of drivers going out. (Hence, the only unknown is the volume of drivers representing the commodity.)

WS. The wait-and-see deterministic reformulation of TAP problems (B-SO) and (B-UE), where vector $\mathbf{b}(\boldsymbol{\xi})$ is defined by the relation 5.5, is written below. Note, that since the right-hand-side vector depends on $\boldsymbol{\xi}$, it implies the feasible flow $\mathbf{f}$, and thus the optimal solution. In WS approach, we wait for the realisation of randomness first, after which we determine the optimal flow. Hence, in this reformulation, we get the feasible set restricted by following constraints for every possible realisation $\boldsymbol{\xi}^s$ as follows

$$\begin{aligned} f_a &= \sum_{i\in\mathcal{C}} x_{ia} && \forall a \in A, \\ \mathbf{A}\mathbf{x}_i &= \mathbf{b}_i(\boldsymbol{\xi}^s) && \forall i \in \mathcal{C}, \\ \mathbf{x}_i &\geq \mathbf{o} && \forall i \in \mathcal{C}. \end{aligned} \tag{5.6}$$

Results taking into account this approach are presented later (with a given distribution of random vector), and compared with here-and-now EV reformulation.

EV. Since the realisation of random vector $\boldsymbol{\xi}$ is unknown in the moment of taking decision, the solution can be "worse" or "better" if we take the decision before. Easily, we use the here-and-now expected value deterministic reformulation by replacing the random vector $\boldsymbol{\xi}$ by its expected value $E\boldsymbol{\xi}$. Then the TAP-constraints are in this deterministic form

$$\begin{aligned} f_a &= \sum_{i\in\mathcal{C}} x_{ia} && \forall a \in A, \\ \mathbf{A}\mathbf{x}_i &= \mathbf{b}_i(E\boldsymbol{\xi}) && \forall i \in \mathcal{C}, \\ \mathbf{x}_i &\geq \mathbf{o} && \forall i \in \mathcal{C}. \end{aligned} \tag{5.7}$$

Note, that the expected value $E\boldsymbol{\xi}$ does not have to be an integer number, and thus the equation cannot be achieved with a integer decision variable. This special case can be solved by rounding the expected value, or taking the smallest integer number not less then the expected value.

Example 5.2 (Comparison of WS and EV results).
Recall the basic TAP problem "One Simple Model" (example 4.2). We consider the randomness appearing in the right-hand-side vector, and in this part we compare WS and

EV approaches. Model solved in GAMS is included in the folder *myTAP3*. Let us just present important results from the attached file *OUT.txt*.

Because of the effect described in foregoing paragraph, and since we have no additional information about the network, we consider a discrete uniformly distributed random vector $\boldsymbol{\xi}$ with expected value $E\boldsymbol{\xi} = 0$ (i.e. $E\boldsymbol{\xi} \in \mathbb{Z}$). From a practical point of view, we generate a number of scenarios $\boldsymbol{\xi}^s$ and compute the expected value as a weighted average (with equal weights for uniform distribution). However, this value $E\boldsymbol{\xi}$ can only rarely be an integer number, we consider in our model the mean as $\lceil E\boldsymbol{\xi} \rceil$ instead.

The right-hand-side vector $\mathbf{b}$ will be affected by random vector (ξ_1, ξ_2) such that for commodity $i \in \mathcal{C}$ the demand is $b_{ij} \pm \xi_i$ for nonzero b_{ij}, where $i \in \mathcal{C}, j \in N$. See the difference between the original deterministic model and stochastic alteration

$$\begin{aligned} \mathbf{b}_1 &= (8, 0, 0, -8), & \mathbf{b}_2 &= (0, 4, -4, 0), \\ \mathbf{b}_1(\boldsymbol{\xi}) &= (8 + \xi_1, 0, 0, -8 - \xi_1), & \mathbf{b}_2(\boldsymbol{\xi}) &= (0, 4 + \xi_2, -4 - \xi_2, 0). \end{aligned}$$

In fact, if we denote $l_i = \max_{j \in N} b_{ij}$, vector $\mathbf{b}_i(\boldsymbol{\xi})$ can be rewritten as

$$\mathbf{b}_i(\lceil E\boldsymbol{\xi} \rceil) = \mathbf{b}_i + \frac{\lceil E\xi_i \rceil}{l} \mathbf{b}_i.$$

Results presented later are for discrete random vector $\boldsymbol{\xi} \sim U_d(-3, 3)$ (i.e. $S = \{-3, -2, \ldots, 3\}$). After generating 200 scenarios, we got these expected values over all scenarios for each commodity

```
Expected Value for commodities in J:       Exi(J) = (-0.290, -0.225)
Expected Value ceiling:              ceil(Exi(J)) = ( 0.000,  0.000)
```

Recall output from the original problem without randomness with optimal flows $\mathbf{f}^*$ for (B-SO) and $\bar{\mathbf{f}}^*$ for (B-UE), respectively.

$$C(\mathbf{f}^*) = z^* = 488, \quad C(\bar{\mathbf{f}}^*) = 493, \quad \rho = 1.01025.$$

See below, that the EV reformulation gives in our model exactly the same solution as the original problem. This is because of distribution with zero mean and a small variance. Look above to compare, that the "real" expected value is zero for neither the first, nor the second commodity. This is enhanced by the revaluation to an integer number. The meaning of `zzUE_EV` is the same as in previous models, the cost of UE-solution.

```
------------------------
here-and-now EV
----------------
  Cost of the Flow:          zSO_EV  =    488.000
                             zzUE_EV =    493.000

  HN Price of Anarchy:       PoA     =    1.01025
```

In opposite to EV, WS approach computes the model again for all different realisations of the random vector. Expected value of price of anarchy ρ_{WS}, and its variation follow

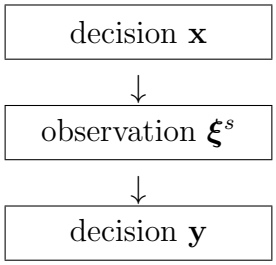
```
--------------------------
wait-and-see
-------------
  Expected value of OS solutions:          478.54
  Expected value of UE solutions:          483.37
  Expected value of PoA:                  1.01250
  Variance of PoA:                        0.00031
```

More details are shown in output file included to the related model in myTAP3 folder.

Here-and-now recourse formulation (RF). The variation of right-hand side can be modelled by modifying the objective function. We add new members expressing the deviation from an "expected solution", and we build so-called *recourse model*.

The recourse model uses so-called soft constraints. It accepts the violation of constraint, but the cost of violation will influence cost of the original solution. The randomness in the constraints can be expressed via new nonnegative variables $y_i^+(\boldsymbol{\xi}), y_i^-(\boldsymbol{\xi})$ that catch out the deviation from the original value $\mathbf{b}_i$.

Since the values of variables y_i^+, y_i^- depend on realisation of the random vector, we get a two-stage program given by the following scheme of decision and observation steps



In order to declare the recourse deterministic reformulation of our stochastic programming problem, denote by $\boldsymbol{\beta}_i$ the following vector

$$\boldsymbol{\beta}_i = \frac{1}{\max\limits_{n \in N} b_{in}} \mathbf{b}_i,$$

Such vector $\boldsymbol{\beta}_i$ is just modified right-hand-side vector $\mathbf{b}_i$ that expresses the unit demand. For further, we call it a *commodity direction*. Hence, its components are zero with exception of that one, which correspond to the source and target (these are $+1$ and -1).

Example 5.3.
For clearness, we present a small example. Suppose demand vector $\mathbf{b}_1$ from our simple example. Then its random variant and implied commodity direction are in the form

$$\mathbf{b}_1(\boldsymbol{\xi}) = \begin{bmatrix} 8+\xi \\ 0 \\ 0 \\ -8-\xi \end{bmatrix}, \qquad \boldsymbol{\beta}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}.$$

Let say, we expect $\mathbf{A}\mathbf{x}_i$ users of the commodity $i \in \mathcal{C}$. The number y_i^+ plays the role of number of users, that came more than expected, i.e. the overflow. (Similarly y_i^- expresses

the number of cars less than expected - the lack of users.) Because of the balance constraint in the model, it must hold the more drivers come in the network, the more must go out at their origin point.

From this point of view, the relaxed constraints are derived as follows

$$\mathbf{A}\mathbf{x}_i + y_i^+(\boldsymbol{\xi})\boldsymbol{\beta}_i - y_i^-(\boldsymbol{\xi})\boldsymbol{\beta}_i = \mathbf{b}_i(\boldsymbol{\xi})$$

where vector $\mathbf{b}_i(\boldsymbol{\xi})$ is given same as above (5.5)

$$\mathbf{b}_i(\boldsymbol{\xi}) := \mathbf{b}_i + \boldsymbol{\xi}\boldsymbol{\beta}_i.$$

For abbreviation, denote $\mathbf{q}$ the vector (q_+^T, q_-^T), and $\mathbf{y}_i(\boldsymbol{\xi})$ the vector $(y_i^+(\boldsymbol{\xi}), y_i^-(\boldsymbol{\xi}))$. The RF approach corresponds to a *two-stage program with recourse* (5.8)

$$\begin{array}{rll} \min\limits_{\mathbf{f},\mathbf{x}} & z(\mathbf{f}) + E_{\boldsymbol{\xi}}Q(\mathbf{x},\boldsymbol{\xi})), & \\ \text{where} & Q(\mathbf{x},\boldsymbol{\xi}) = \min\limits_{\mathbf{y}(\boldsymbol{\xi})} \sum\limits_{i\in\mathcal{C}} \mathbf{q}^T\mathbf{y}_i(\boldsymbol{\xi}) & \\ & \qquad \text{s.t.} \quad \mathbf{W}\mathbf{y}_i(\boldsymbol{\xi})\,[\boldsymbol{\beta}_i, \boldsymbol{\beta}_i] = \mathbf{b}_i(\boldsymbol{\xi}) - \mathbf{A}\mathbf{x}_i \quad \forall i \in \mathcal{C} & \\ & \qquad\quad\ \ \mathbf{y}_i(\boldsymbol{\xi}) \geq \mathbf{o} \quad \forall i \in \mathcal{C} & \end{array} \tag{5.8}$$

In the simplest case, we may just penalise the deviation in the constraints by penalty coefficients q_+, q_-, which are constant. The model (5.8) is said to be with *simple recourse*, if the matrix $\mathbf{W}$ is in the form $(\mathbf{I}, -\mathbf{I})$. Such simple recourse we will take into account in our next considerations, i.e. $\mathbf{W} = \mathrm{diag}(1, -1)$.

In fact, we want to do a decision at the beginning, that will be at least penalised after the observation of the random vector $\boldsymbol{\xi}$. In other words, we want to minimize the deviation from original right-hand-side vector $\mathbf{b}_i$.

Then the deterministic reformulation of SO two-stage program with recourse is as follows

$$\begin{array}{lll} \min & \sum\limits_{a\in A} c_a(f_a)f_a + \sum\limits_{s\in S} p_s \sum\limits_{i\in\mathcal{C}} \mathbf{q}^T\mathbf{y}_i(\boldsymbol{\xi}^s) & \\ \text{s.t.} & f_a = \sum\limits_{i\in\mathcal{C}} x_{ia} & \forall a \in A \\ & y_i^+(\boldsymbol{\xi}^s)\boldsymbol{\beta}_i - y_i^-(\boldsymbol{\xi}^s)\boldsymbol{\beta}_i = \mathbf{b}_i(\boldsymbol{\xi}^s) - \mathbf{A}\mathbf{x}_i & \forall i \in \mathcal{C} \\ & \mathbf{y}_i(\boldsymbol{\xi}^s) \geq \mathbf{o} & \forall i \in \mathcal{C} \\ & \mathbf{x}_i \geq \mathbf{o} & \forall i \in \mathcal{C} \end{array} \tag{5.9}$$

In our model implemented in GAMS (see document *myTAP4.gms*) we used different penalisation coefficients q_+, q_-. Since the values y_i^+, y_i^- express how many more, respectively how many less drivers came than expected, we may set the penalisation q_- to zero, or to negative number. Meaning of this assumption results from nature: less drivers make the system better, but more drivers congest the traffic, and this should be penalised.

A primal approximation of price per each extra user was done by the average travel cost of one unit from the original deterministic example 4.2, i.e. $\approx \frac{488}{12}$. Since cost function are increasing and nonlinear, we may involve additional costs higher that expected. Hereby, $q_+ = 70$ and $q_- = -20$ are conservative estimates.

```
================================================================================
= Recourse coefficients:
=             qplus =   70.0 , qminus =  -20.0
================================================

SO Optimal flow x:                          UE Optimal flow x:
------------------                          ------------------
          com1       com2           |              com1       com2
                                     |
arc1      3.00       0.00           |    arc1      3.00       0.00
arc2      0.00       2.00           |    arc2      0.00       3.00
arc3      3.00       0.00           |    arc3      4.00       0.00
arc4      0.00       2.00           |    arc4      0.00       3.00
arc5      3.00       1.00           |    arc5      3.00       2.00
arc6      3.00       0.00           |    arc6      4.00       0.00
arc7      0.00       1.00           |    arc7      0.00       2.00
                                     |
SO Solution:                         |    UE Solution:
------------                         |    ------------
obj. function   zSO_min =  477.70    |    obj.function    zUE_min =  349.20
cost of flow       zzSO =  301.00    |    cost of flow       zzUE =  474.00

  Price of Anarchy:
  -----------------
  poa = 1.574751
```

In RF model, we can easily change penalisation to achieve solution with different interpretations. By setting the price for overloading the network much higher (than for underloading), we almost simulate the "worst case" scenario. The behaviour of the mathematical program is shown below. Hence, our stochastic TAP finds more efficient to expect traffic on higher volume.

```
================================================================================
= Recourse coefficients:
=             qplus =  200.0 , qminus =  -20.0
================================================

SO Optimal flow x:                          UE Optimal flow x:
------------------                          ------------------
          com1       com2           |              com1       com2
                                     |
arc1      4.00       0.00           |    arc1      4.00       0.00
arc2      0.00       3.00           |    arc2      0.00       4.00
arc3      4.00       0.00           |    arc3      5.00       0.00
arc4      0.00       3.00           |    arc4      0.00       4.00
arc5      4.00       2.00           |    arc5      4.00       2.00
arc6      4.00       0.00           |    arc6      5.00       0.00
arc7      0.00       2.00           |    arc7      0.00       2.00
                                     |
                                     |
```

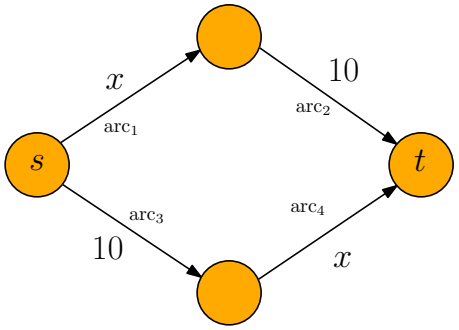
```
SO Solution:                          |    UE Solution:
------------                          |    ------------
obj. function   zSO_min =  666.20     |    obj.function    zUE_min =  415.30
cost of flow       zzSO =  553.00     |    cost of flow       zzUE =  701.00

  Price of Anarchy:
  -----------------
  poa = 1.267631
```

6. Network Design

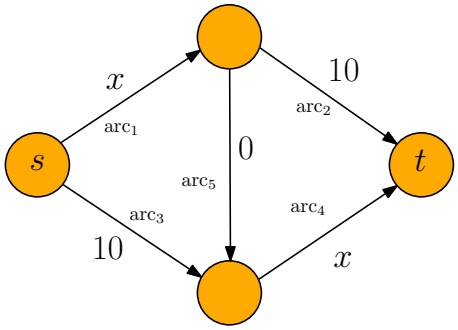
6.1. Braess Paradox and Related Problems

One can presume, that the flow becomes more efficient if more arcs are added in the network. Braess [5] discovered in 1968 an example, where this is denied. Since the time the example was presented, many interesting results were reached in this part. In this section we introduce a few of them, and show how difficult is to find such paradox in a given network.

Example 6.1 (Braess).
In 1968 Braess presented a simple example of network with four nodes, where the cost of UE-flow becomes worse, when a new arc is added (see the network instance below). Suppose, there are 10 drivers travelling between two nodes s and t. In the first case, both paths have identical cost functions and the flow is uniformly divided into two possible directions. The second network describes the effect of Braess paradox - noncooperative behaviour of the users makes the total cost higher. Although in first graph, both SO-



Network without Braess paradox



Network affected by paradox

and UE-flows are equal $\mathbf{f} = (5, 5, 5, 5)$, by connecting two intermediate nodes with a zero cost arc, the system optimum remains the same while user equilibrium moves to $\bar{\mathbf{f}} = (10, 0, 0, 10, 10)$. (Compare particular paths to prove.) Easily, by adding a new arc, the cost $C(\mathbf{f}) = 150$ increased to $C(\bar{\mathbf{f}}) = 200$.

Braess ratio. Roughgarden [19] in his paper proved results, that throw the problem in complete another light. We try to summarize the most important of them.

Definition 6.1 (Braess ratio).
Suppose a single-commodity TAP on a graph $G = (N, A)$, with demand vector $\mathbf{b}$ and cost functions $(c_a)_{a\in A}$. The *Braess ratio* $\beta = \beta(G, \mathbf{b}, (c_a)_{a\in A})$ of a single-commodity TAP is defined by

$$\beta = \max_{H\subseteq G} \frac{C(\mathbf{f})}{C(\mathbf{f}_H)},$$

where H is a subgraph of graph G that contains an $s - t$ path, and $\mathbf{f}$ and $\mathbf{f}_H$ are user equilibria on G and H, respectively.

There are various ways how to define the Braess ratio for a multicommodity networks. This will not be necessary for us now, because it makes things much more complicated.

Roughgarden proved a condition for boundary of the Braess ratio in a single-commodity network.

LEMMA 6.1 ([19]).
For every single commodity TAP the following holds

$$\beta \leq \rho.$$

In Braess's example, the Braess ratio is $\frac{4}{3}$. In a preceding section, we showed, that the upper bound for price ρ is given by the Pigou bound α, which is for linear cost functions also $\frac{4}{3}$. Since $\beta = \alpha$, this Pigou bound is tight.

LEMMA 6.2 ([19]).
For every single-commodity TAP with $n \geq 2$ nodes, the Braess ratio is limited by following boundary

$$\beta \leq \left\lfloor \frac{n}{2} \right\rfloor.$$

One naturally asks: given a network, is it affected by Braess paradox? If so, how to detect it? A big progress in this uncertain question was made for linear cost function, which usually illustrate "the simplest case". The task can be also formulated as follows: Given a single-commodity TAP with linear cost functions, find a subgraph, that minimizes the cost of user equilibrium. In literature is this problem usually called *Linear Network Design (LND).*

The situation is well expressed through following theorem. Denote a γ-approximation algorithm an algorithm, that returns in polynomial time a solution no more than γ times as costly as an optimal solution. An algorithm that for a network G returns the whole network is called *trivial algorithm.* As we showed above, the trivial algorithm in LND is $\frac{4}{3}$-approximation algorithm.

THEOREM 6.1 ([19]).
For every $\epsilon > 0$ there exists no $\left(\frac{4}{3} - \epsilon\right)$-approximation algorithm for Linear Network Design.

A similar theorem can be proved for a General Network Design problem, and thus finding the Braess paradox in a general network is not solvable without using full enumeration method.

R-closure of the Network

Recall, we used to write $\operatorname{argmin}_{\mathbf{x}}\{f(\mathbf{x}) | \mathbf{x} \in X\}$ in the meaning of set of all optimal solutions $\mathbf{x} \in X$, that minimize the objective function $f(\mathbf{x})$ (see chapter 3).

The main point of TAP problem can be also stated as finding the minimum of price of anarchy ρ. Denote $\mathbf{f}^*, \bar{\mathbf{f}}^*$ the optimal flows solving the (B-SO) and (B-UE), respectively. For a given network (with fixed nodes and arcs), the minimization of price of anarchy is the same as the minimization of the cost of the flow $\bar{\mathbf{f}}^*$

$$\min \rho = \min_{\bar{\mathbf{f}}^*} \frac{C(\bar{\mathbf{f}}^*)}{C(\mathbf{f}^*)} \qquad \Longleftrightarrow \qquad \min C(\bar{\mathbf{f}}^*).$$

Note, that the cost of UE solution for a flow $\bar{\mathbf{f}}^* = (\bar{f}_a^*)_{a\in A}$ is computed by

$$\begin{aligned} C(\bar{\mathbf{f}}^*) = \quad & \sum_{a\in A} c_a(\bar{f}_a^*)\bar{f}_a^* \\ \text{where} \quad & \bar{\mathbf{f}}^* \in \ \operatorname{argmin}_{\bar{\mathbf{f}}} \sum_{a\in A} \int_0^{\bar{f}_a} c_a(s)\ \mathrm{d}s \\ & \bar{f}_a = \sum_{i\in\mathcal{C}} x_{ia} && \forall a \in A, \\ & \mathbf{A}\mathbf{x}_i = \mathbf{b}_i && \forall i \in \mathcal{C}, \\ & \mathbf{x}_i \geq \mathbf{o} && \forall i \in \mathcal{C}. \end{aligned} \tag{6.1}$$

Thus, for a fixed network, there is no variable to minimize, because the user equilibrium always exists and is unique (see theorem 4.1).

As we have seen, the way how to find the Braess paradox in a general network is very difficult - the only possible way is full enumeration of all solutions. In order to avoid these trouble, we formulate an another pattern. While, solving the Braess paradox may generally lead to removing one, two, or arbitrary many arcs from the graph, we try to deal with a fixed number of arcs that should be "deleted" from the network.

Let say, we add a new constraint, that guarantees us how many arcs should be token away. Denote the required number of removed arcs by R. We call this additional condition the *R-closure* of the network. The task to minimize the price of anarchy, means to minimize the flow cost, where the UE-flow is constrained by additional constraints on R-closure

$$\begin{aligned} \min_{\boldsymbol{\delta},\bar{\mathbf{f}}} \quad & C(\bar{\mathbf{f}}) \\ \text{s.t.} \quad & \bar{f}_a \leq \delta_a M && \forall a \in A, \\ & \textstyle\sum_{a\in A}(1-\delta_a) = R, \\ & \delta_a \in \{0,1\} && \forall a \in A. \end{aligned} \tag{6.2}$$

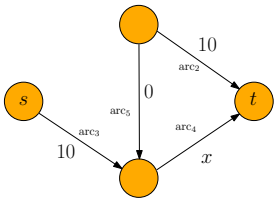
Here, δ_a is a binary variable, that takes the value 0 if arc a is removed, and value 1 if not. Since flow through removed arc must be zero, this is achieved with the first constraint, where M is a "big number" (with respect to all possible flows). The sum over all closed arcs must be equal to $R \in \mathbb{Z}^+$, because of the R-closure.

The symbol $\min_{\boldsymbol{\delta},\bar{\mathbf{f}}}$ denotes in fact $\min_{\boldsymbol{\delta}} \min_{\bar{\mathbf{f}}}$. But since in the cost $C(\bar{\mathbf{f}})$ there is no one member containing δ-variables explicitly, we understand this as a minimizing only over the flow $\bar{\mathbf{f}}$. Thus, the resulting flow is a function of $\boldsymbol{\delta} = (\delta_a)_{a\in A}$, and we want to find that one with minimum objective function value. We perform a simple example of 1-closure below.

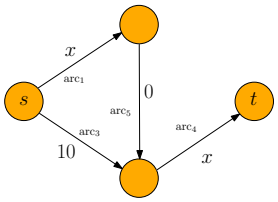
Demonstration of the R-closure

Example 6.2.
Consider the Braess example of a simple network presented in the beginning of these section, with four nodes and five arcs. Let us compute the optimal solution in such network with 1-closure (i.e. removing exactly one arc from the network). By closing one arc, we get solutions as follows:

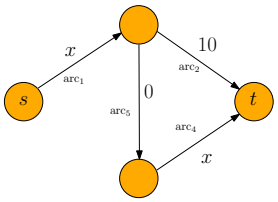
Obviously, closing the arc_5 makes the cost the lowest possible, and thus this is the solution we get.



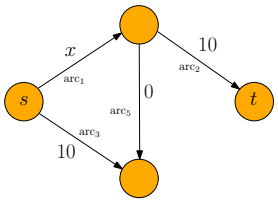
$\boldsymbol{\delta} = (1, 0, 0, 0, 0)$
$\overline{\mathbf{f}}^* = (0, 0, 0, 10, 10)$,
$C(\overline{\mathbf{f}}^*) = 200$.



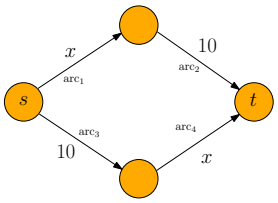
$\boldsymbol{\delta} = (0, 1, 0, 0, 0)$
$\overline{\mathbf{f}}^* = (10, 0, 0, 10, 10)$,
$C(\overline{\mathbf{f}}^*) = 200$.



$\boldsymbol{\delta} = (0, 0, 1, 0, 0)$
$\overline{\mathbf{f}}^* = (10, 0, 0, 10, 10)$,
$C(\overline{\mathbf{f}}^*) = 200$.



$\boldsymbol{\delta} = (0, 0, 0, 1, 0)$
$\overline{\mathbf{f}}^* = (10, 10, 0, 0, 0)$,
$C(\overline{\mathbf{f}}^*) = 200$.



$\boldsymbol{\delta} = (0, 0, 0, 0, 1)$
$\overline{\mathbf{f}}^* = (5, 5, 5, 5, 0)$,
$C(\overline{\mathbf{f}}^*) = 150$.

6.2. Bilevel Programming

Bilevel programming problems (BP) are mathematical programs, where some of the decision variables are solutions of another mathematical programs. The applications can be found in many branches dealing with optimization, like economy, game theory, or other hierarchical optimization opportunities.

Bilevel programming problem is hierarchical, which means that its constraints are defined in part by a second optimization problem.

Let us first introduce the *follower's* (or *lower*) *problem*. This is an mathematical program in a form

$$\begin{array}{rl} \min_x & \mathbf{f}(x, y) \\ \text{s.t.} & \mathbf{g}(x, y) \leq \mathbf{o} \\ & \mathbf{h}(x, y) = \mathbf{o} \end{array} \tag{6.3}$$

where all $\mathbf{f}, \mathbf{g}, \mathbf{h}$ are vector functions. Let us denote $\Psi(y)$ the solution of problem (6.3) for a fixed $y \in \mathbb{R}^p$.

The *leader's* (or *upper*) *problem* is defined as follows

$$\begin{array}{rl} \min_y & F(x(y), y) \\ \text{s.t.} & G(x(y), y) \leq 0 \\ & H(x(y), y) = 0 \\ & x(y) \in \Psi(y) \end{array} \tag{6.4}$$

All functions that occur in the leader's problem (6.4) are said to be *upper functions* (upper obj. function, upper constraints), and functions from the follower's problem (6.3) are said to be *lower functions*.

As we see, the follower's problem solution is given by variable x, which is function of the second variable, i.e. $x = x(y)$. The "inner" problem is in fact parametric mathematical program, where $x(y) \in \Psi(y)$ and $\Psi(y)$ denotes the solution of the problem with a parameter y, i.e. $\Psi(y) = \text{argmin}_z\{f(z, y) | g(z, y) \leq 0, h(z, y) = 0\}$.

The aim of BP is to minimize the lower objective function over the feasible set, after that the upper problem is solved over the resulting set of optimal solutions. Many authors have proved the $\mathcal{NP}$-hardness of bilevel programming. Moreover, in an arbitrary bilevel programming problem the solution existence cannot be guaranteed even if all function are continuous and bounded.

Solution methods. One possible way how to deal with a BP is to derive the description of the function $x(y)$, which can be inserted into the upper problem. Second often used approach is to replace the lower problem by Karush-Kuhn-Tucker optimality conditions (see [3]). This results in a one-level *mathematical program with equilibrium constraints* (MPEC). Several other approaches are possible, and some of them can be found in [11].

6.3. Bilevel Reformulation of R-closure

As presented in the section 6, we developed a new mathematical programming problem (6.2) solving the optimal R-closure of the network. This problem is in fact a bilevel program in the following form

$$\begin{aligned}
\min_{\boldsymbol{\delta}} \quad & \sum_{a\in A} c_a(f_a^*(\delta_a))f_a^*(\delta_a) & & \\
\text{s.t.} \quad & f_a^*(\delta_a) \in \operatorname{argmin}_{f_a} \sum_{a\in A} \int_0^{f_a} c_a(s)\,\mathrm{d}s & & \\
& f_a = \sum_{i\in\mathcal{C}} x_{ia} & \forall a \in A, & \\
& \mathbf{A}\mathbf{x}_i = \mathbf{b}_i & \forall i \in \mathcal{C}, & \\
& \mathbf{x}_i \geq \mathbf{o} & \forall i \in \mathcal{C}, & \\
& \textstyle\sum_{a\in A}(1-\delta_a) = R, & & \\
& f_a \leq \delta_a M & \forall a \in A, & \\
& \delta_a \in \{0,1\} & \forall a \in A. &
\end{aligned} \tag{6.5}$$

The simplest way to solve such problem is a full enumeration of all results (if this is possible with respect to variable types). We used this approach just to compare further solutions. Results reached by full enumeration of 1- and 2-closure in the example 4.2 are as follows (see included file *myTAP5 enumer.gms*)

```
----------------------Traffic Assignment Problem: myTAP5 enumer---------------
Cost of the Flow:

  closed arc
   1               objout =    635.00                (objin =    372.00)
   2               objout =    575.00                (objin =    343.50)
   3               objout =    594.00                (objin =    352.00)
   4               objout =    534.00                (objin =    346.00)
   5               objout =    735.00                (objin =    431.50)
   6               objout =    708.00                (objin =    438.00)
   7               objout =    503.00                (objin =    327.50)


  closed arcs
```

```
1, 2            objout =    651.00              (objin =    379.50)
1, 3            objout =    836.00              (objin =    452.00)
1, 4            objout =    708.00              (objin =    420.00)
1, 5            objout =    735.00              (objin =    431.50)
1, 6            objout =    735.00              (objin =    431.50)
1, 7            objout =    735.00              (objin =    431.50)
2, 3            objout =    635.00              (objin =    366.50)
2, 4            objout =    600.00              (objin =    369.00)
2, 5            objout =    600.00              (objin =    369.00)
2, 6            objout =    980.00              (objin =    564.00)
2, 7            objout =    980.00              (objin =    564.00)
3, 4            objout =    980.00              (objin =    564.00)
3, 5            objout =   1176.00              (objin =    624.00)
3, 6            objout =    744.00              (objin =    448.00)
3, 7            objout =    651.00              (objin =    386.50)
4, 5            objout =    840.00              (objin =    504.00)
4, 6            objout =    728.00              (objin =    456.00)
4, 7            objout =    560.00              (objin =    368.00)
5, 6            objout =    560.00              (objin =    368.00)
5, 7            objout =    735.00              (objin =    431.50)
6, 7            objout =    708.00              (objin =    438.00)

.
.
.
etc.
```

Since the bilevel program (6.5) is a very special mathematical problem, we get the solution by a tricky adjustments. From the nature of problem, we can reformulate finding the R-closure in a network to a sequence of subproblems solving much easier tasks.

The leader wants to minimize the cost of a flow, and thus to find one of possible solution closest to the SO-optimal one. Obviously, a pair $(\mathbf{f}^*, \boldsymbol{\delta}^*)$ is a solution to program (6.5) if and only if $\boldsymbol{\delta}^*$ is the solution to (B-SO) with a R-closure constraints. For a given $\boldsymbol{\delta}^*$, users in the network spread to all remaining arcs to follow their UE-solution. Hence

1. for a given R, find the best R-closure $\boldsymbol{\delta}^*$ of the (B-SO) problem,
2. for $\boldsymbol{\delta}^*$ given by 1., decide the flow $\mathbf{f}^*$ solving the UE follower's problem,
3. compute the cost $C(\mathbf{f}^*)$.

Compare results from full enumeration showed above and R-closures found by described algorithm. Problem implemented in GAMS software is to be found under name *myTAP5*. See the output file for $R = 1, 2, 3, 4$ below (for $R > 4$ the problem becomes infeasible because of not existing path from origin to destination)

```
----------------------Traffic Assignment Problem: myTAP5----------------------

Cost of the Flow:
  R =  1
  ------------------
  objout   =  503.00
  objin    =  327.50
  delta(I) =  ( 1, 1, 1, 1, 1, 1, 0)

  R =  2
  ------------------
  objout   =  560.00
  objin    =  368.00
  delta(I) =  ( 1, 1, 1, 0, 1, 1, 0)

  R =  3
  ------------------
  objout   =  708.00
  objin    =  420.00
  delta(I) =  ( 0, 0, 1, 0, 1, 1, 1)

  R =  4
  ------------------
  objout   =  840.00
  objin    =  504.00
  delta(I) =  ( 0, 1, 1, 0, 0, 1, 0)
```

6.4. Street Cleaning Problem

Having solved the problem of optimal closure of the network, we may formulate a more complete pattern with working title *Street Cleaning Problem* (SCP).

Suppose, there are available R street cleaning vehicles, which clean the streets of our network. They do it every day unless all streets have not been cleaned. If a street (or an arc) is cleaned, the traffic must deflected to the other streets. Since static TAP itself is linked to a specific time period in which the number of drivers is constant, we suppose that one cleaning vehicle takes it exactly one period to clean one street.

The objective of SCP is to find the right schedule of street cleaning, such that we use the given number of cleaning vehicles and the total time is minimized. Hereby we mean, find a sequence of R-closures, such that at the end each arc has been closed exactly once, and the sum of cost of all flows is minimized.

As a simplification, we consider that $|A| = \tau R$, where $\tau \in \mathbb{Z}, \tau \geq 2$, and where $|A|$ denotes the number of arcs in a network. Further, we must take into account, that for each closure of the optimal sequence the flow must be feasible, i.e. for each commodity there must always exist a path from its source to its target. Cases of 1- and $|A|$-closures are trivial[1], and we will not consider it.

[1]For a solution of SCP within 1-closures we must close exactly one arc in $|A|$ periods. On the other hand, $|A|$-closure is never feasible, because there exists no path between origin and destination.

Greedy approach. One of possible solution methods is "greedy algorithm", i.e. compute the R-closure with the least cost, then find the cheapest R-closure for δ_a that has not been used yet, etc. Although such algorithm seems to be reasonable, none algorithm of the huge class of greedy algorithm needs to achieve the optimal value. On the other hand, due to its simplicity, its computing demands are much lower than for its "precise" version shown further.

Bilevel formulation of SCP. With respect to our assumption, denote T the set of all time periods, i.e. $T = \{1, \ldots, \tau\}$. SCP in symbolic form written as a bilevel programming problem is the following integer mathematical program with binary variables

$$\begin{aligned}
\min_{\boldsymbol{\delta}} \quad & \sum_{t\in T}\sum_{a\in A} c_a(f^*_{ta}(\delta_{ta}))f^*_{ta}(\delta_{ta}) \\
\text{s.t.} \quad & f^*_{ta}(\delta_{ta}) \in \operatorname{argmin}_{f_{ta}} \sum_{a\in A} \int_0^{f_{ta}} c_a(s)\,\mathrm{d}s \\
& f_{ta} = \sum_{i\in\mathcal{C}} x_{tia} && \forall a \in A, \forall t \in T, \\
& \mathbf{A}\mathbf{x}_{ti} = \mathbf{b}_{ti} && \forall i \in \mathcal{C}, \forall t \in T, \\
& \mathbf{x}_{ti} \geq \mathbf{o} && \forall i \in \mathcal{C}, \forall t \in T, \\
& \textstyle\sum_{a\in A}(1-\delta_{ta}) = R && \forall t \in T, \\
& f_{ta} \leq \delta_{ta} M && \forall a \in A, \forall t \in T, \\
& \delta_{ta} \in \{0,1\} && \forall a \in A, \forall t \in T. \\
& \boldsymbol{\delta}_t^T \boldsymbol{\delta}_r = |A| - 2R && \forall t, r \in T, t \neq r
\end{aligned} \tag{6.6}$$

The last condition ensures us the disparity of all R-closures, and thus the desired property is ensured, that none street can be cleaned more times.

With respect to solution concept mentioned already in the previous section, we may divide this huge bilevel problem into smaller NLP subproblems, that lead to finding a best sequence of R-closures with respect to system optimum. Hence, we first concentrate on finding the best sequence of closures for system optimum problem. According to page 58, now we solve the following mathematical program

$$\begin{aligned}
\min_{\boldsymbol{\delta}} \quad & \sum_{t\in T}\sum_{a\in A} c_a(f_{ta}(\delta_{ta}))f_{ta}(\delta_{ta}) \\
\text{s.t.} \quad & f_{ta} = \sum_{i\in\mathcal{C}} x_{tia} && \forall a \in A, \forall t \in T, \\
& \mathbf{A}\mathbf{x}_{ti} = \mathbf{b}_{ti} && \forall i \in \mathcal{C}, \forall t \in T, \\
& \mathbf{x}_{ti} \geq \mathbf{o} && \forall i \in \mathcal{C}, \forall t \in T, \\
& \textstyle\sum_{a\in A}(1-\delta_{ta}) = R && \forall t \in T, \\
& f_{ta} \leq \delta_{ta} M && \forall a \in A, \forall t \in T, \\
& \delta_{ta} \in \{0,1\} && \forall a \in A, \forall t \in T. \\
& \boldsymbol{\delta}_t^T \boldsymbol{\delta}_r = |A| - 2R && \forall t, r \in T, t \neq r
\end{aligned} \tag{6.7}$$

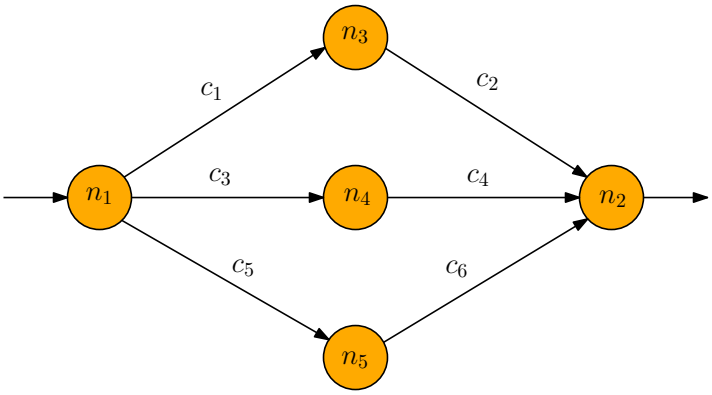
The searched sequence of closures we observe as solution of (6.7).

Example 6.3.
As we mentioned above, the case of 1-closures is trivial, and we will concentrate on less clear tasks. In order to keep the verity of equality $|A| = \tau R = |T|R$ also for other closures, we cannot show the SCP solution on our simple example with 7 arcs. Due to this fact, we

consider a new graph given as below. Same as before, we suppose linear cost functions. Concrete values and functions are given as follows

$$c_1 = 8 + 3f_1 \quad c_2 = 8 + 4f_2 \quad c_3 = 10 + 2f_3$$
$$c_4 = 2 + 5f_4 \quad c_5 = 6 + 4f_5 \quad c_6 = 2 + 5f_6$$

We assume the number of 20 drivers travelling from node n_1 to n_2.



An SCP model

Obviously, in this special graph the solution of 2-closures sequence will always include such closures of arcs, that compose a path. Truthfulness of this statement is supported by our results. Problem implemented in GAMS is attached as *myTAP6 scp*, we just present our output file containing all important solutions.

```
------------------Traffic Assignment Problem: myTAP6 SCP---------------------

  t = 1       delta(I) =  ( 0, 0, 1, 1, 1, 1)
  t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
  t = 3       delta(I) =  ( 1, 1, 0, 0, 1, 1)

  Solution:
         objout   = 5284.00
         objin    = 3008.00
```

As mentioned, another approach is to use an greedy algorithm. By full enumeration we get following results for any 2-closures (see folder *myTAP6 scp enumeration* for closer view)

```
----------------------Traffic Assignment Problem: myTAP6 SCP enumer----------

Full enumeration:

  closed arcs
  -----------
   1, 2            objout =   1780.0
   1, 3            objout =   3760.0
   1, 4            objout =   3760.0
   1, 5            objout =   3040.0
   1, 6            objout =   3040.0
   2, 3            objout =   3760.0
```

```
2, 4            objout =   3760.0
2, 5            objout =   3040.0
2, 6            objout =   3040.0
3, 4            objout =   1824.0
3, 5            objout =   3120.0
3, 6            objout =   3120.0
4, 5            objout =   3120.0
4, 6            objout =   3120.0
5, 6            objout =   1680.0
```

First, choose the cheapest closure, i.e. the closure (5,6). After that, find the second cheapest closure. If it contains already used arc, then discard it. The second cheapest is closure (1,2), and this forces the choice of the last one (3,4) irrespective of its cost. Total cost of all flows in the sequence is 5284 as before. Thus in this case, our greedy algorithm has found optimal solution, which is not always guaranteed.

Stochastic Street Cleaning Problem

Realize, that solution found in example 6.3 will remain optimal for any permutation of closures. As we said, SCP (as multiple TAP problem) is linked to specific time period in which the traffic volume remains constant. So far, primal assumption of problem 6.6 is, that the vector $\mathbf{b}_i$ has to be the same for every commodity in all period.

Now, we allow deviation in the number of drivers, which corresponds to stochastic SCP formulation. In order to simulate a real situation, let the deviation be expressed by a randomness. Moreover, let random vector $\boldsymbol{\xi}_t$ be associated to a specific time interval in which we assume constant flow. Hereby, we can model the changes of flow in time, such as rush hour or common traffic.

Let random vector $\boldsymbol{\xi}_t$ has known probability distribution for every $t \in T$. As before, we suppose $|A| = |T|R$. Stochastic SCP with randomness appearing in right-hand-side vector is formally written as following stochastic program

$$
\begin{aligned}
\min_{\boldsymbol{\delta}} \quad & \sum_{t\in T}\sum_{a\in A} c_a(f^*_{ta}(\delta_{ta}))f^*_{ta}(\delta_{ta}) \\
\text{s.t.} \quad & f^*_{ta}(\delta_{ta}) \in \operatorname{argmin}_{f_{ta}} \sum_{a\in A} \int_0^{f_{ta}} c_a(s)\,\mathrm{d}s \\
& f_{ta} = \sum_{i\in\mathcal{C}} x_{tia} && \forall a \in A, \forall t \in T, \\
& \mathbf{A}\mathbf{x}_{ti} = \mathbf{b}_{ti}(\boldsymbol{\xi}_t) && \forall i \in \mathcal{C}, \forall t \in T, \\
& \mathbf{x}_{ti} \geq \mathbf{o} && \forall i \in \mathcal{C}, \forall t \in T, \\
& \textstyle\sum_{a\in A}(1-\delta_{ta}) = R && \forall t \in T, \\
& f_{ta} \leq \delta_{ta} M && \forall a \in A, \forall t \in T, \\
& \delta_{ta} \in \{0,1\} && \forall a \in A, \forall t \in T. \\
& \boldsymbol{\delta}_t^T \boldsymbol{\delta}_r = |A| - 2R && \forall t, r \in T, t \neq r
\end{aligned}
\tag{6.8}
$$

As presented in chapter 5, we may reformulate this problem in various ways. Differences between here-and-now and wait-and-see reformulations are given below.

Remark. Note, that for a specific random vectors $\boldsymbol{\xi}_t, t \in T$, solution of problem 6.8 will be unique for any permutation, and hereby closures must be made in a given order.

Our computation are made on preceding SCP-example with 80 users and following discrete probability distributions

$$\boldsymbol{\xi}_1 \sim U_d(-30, 0), \qquad \boldsymbol{\xi}_2 \sim U_d(-20, 20), \qquad \boldsymbol{\xi}_3 \sim U_d(-40, -10).$$

WS approach. In wait-and-see deterministic reformulation, we solve problem 6.8 separately for each realisation of randomness. In our case, for every sequence of realisations $(\boldsymbol{\xi}_1^s, \ldots, \boldsymbol{\xi}_{|T|}^s)$. Hereby, we assume the optimal closure to be function of random vector, i.e. $\boldsymbol{\delta} = \boldsymbol{\delta}(\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_{|T|})$. Formally rewritten, for a specific realisation, we just replace $\boldsymbol{\xi}_t$ by $\boldsymbol{\xi}_t^s$ in program 6.8.

According to discussion in section 6.3, solution of 6.8 is given by solution of closure in the outer problem

$$\begin{array}{lll}
\min_{\boldsymbol{\delta}} & \sum_{t\in T}\sum_{a\in A} c_a(f_{ta}(\delta_{ta}))f_{ta}(\delta_{ta}) & \\
\text{s.t.} & f_{ta} = \sum_{i\in\mathcal{C}} x_{tia} & \forall a \in A, \forall t \in T, \\
& \mathbf{A}\mathbf{x}_{ti} = \mathbf{b}_{ti}(\boldsymbol{\xi}_t^s) & \forall i \in \mathcal{C}, \forall t \in T, \\
& \mathbf{x}_{ti} \geq \mathbf{o} & \forall i \in \mathcal{C}, \forall t \in T, \\
& \sum_{a\in A}(1-\delta_{ta}) = R & \forall t \in T, \\
& f_{ta} \leq \delta_{ta} M & \forall a \in A, \forall t \in T, \\
& \delta_{ta} \in \{0, 1\} & \forall a \in A, \forall t \in T. \\
& \boldsymbol{\delta}_t^T \boldsymbol{\delta}_r = |A| - 2R & \forall t, r \in T, t \neq r
\end{array} \tag{6.9}$$

Implementation of WS approach on problem 6.8 is attached in folder *myTAP7*. For number of scenarios (for each period $t \in T$) we get optimal closures. Wait-and-see is from the nature the best response to the randomness. We present just optimal solutions for 10 randomly generated scenarios, where `objout` is the total cost of SCP solution.

```
----------------------Traffic Assignment Problem: myTAP7------------------


  --------------------------------
  wait-and-see EV
  ---------------
  ||No. 1||
  t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
  t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
  t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)
        objout   =  60182.00
        objin    =  31373.00

  ||No. 2||
  t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
  t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
  t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)
        objout   =  59327.00
        objin    =  30971.50

  ||No. 3||
```

```
t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)
       objout   =  63027.00
       objin    =  32863.50

||No. 4||
t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)
       objout   =  52778.00
       objin    =  27619.00

||No. 5||
t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)
       objout   =  43986.00
       objin    =  23111.00

||No. 6||
t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)
       objout   =  57443.00
       objin    =  30003.50

||No. 7||
t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)
       objout   =  40279.00
       objin    =  21213.50

||No. 8||
t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)
       objout   =  57018.00
       objin    =  29759.00

||No. 9||
t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)
       objout   =  55791.00
       objin    =  29155.50

||No.10||
t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
```

```
t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)
       objout   =  49239.00
       objin    =  25811.50
```

We see, that in our case for each scenario with known distributions the optimal closure is the same. This is not a rule, and for an another network the solution may differ within every scenario.

HN Expected value. As the first here-and-now approach, we conduct the EV approach already mentioned on page 46. This approach consists in replacing the randomness by its expected value. Hereby, we make a deterministic reformulation in sense of EV by simple rewrite the demand condition into

$$\mathbf{A}\mathbf{x}_{ti} = \mathbf{b}_{ti}(E\boldsymbol{\xi}_t) \qquad \forall i \in \mathcal{C}.$$

Recall, that expected value $E\boldsymbol{\xi}_t$ is assigned to specific time interval, i.e. to the t-th period. According to results from the previous part, clearly we can write program solving the best closure sequence as follows

$$\begin{array}{lll} \min_{\boldsymbol{\delta}} & \sum_{t\in T}\sum_{a\in A} c_a(f_{ta}(\delta_{ta}))f_{ta}(\delta_{ta}) & \\ \text{s.t.} & f_{ta} = \sum_{i\in\mathcal{C}} x_{tia} & \forall a \in A, \forall t \in T, \\ & \mathbf{A}\mathbf{x}_{ti} = \mathbf{b}_{ti}(E\boldsymbol{\xi}_t) & \forall i \in \mathcal{C}, \forall t \in T, \\ & \mathbf{x}_{ti} \geq \mathbf{o} & \forall i \in \mathcal{C}, \forall t \in T, \\ & \sum_{a\in A}(1-\delta_{ta}) = R & \forall t \in T, \\ & f_{ta} \leq \delta_{ta} M & \forall a \in A, \forall t \in T, \\ & \delta_{ta} \in \{0,1\} & \forall a \in A, \forall t \in T. \\ & \boldsymbol{\delta}_t^T \boldsymbol{\delta}_r = |A| - 2R & \forall t, r \in T, t \neq r \end{array} \tag{6.10}$$

Similarly to a general TAP reformulation solved in previous parts, the expected value $E\boldsymbol{\xi}_t$ does not need to be an integer number. One can round this value, or (as we did) take the value $\lceil E\boldsymbol{\xi}_t \rceil$. In this way, we suppose "worse case" instead of ordinary rounding.

Solution and implementation in software GAMS is part of file *myTAP7.gms*. Here we use the same randomly generated scenarios as in WS approach presented before. See reached HN optimum below

```
----------------------Traffic Assignment Problem: myTAP7-------------------

  --------------------------------
  here-and-now EV
  ---------------
  Expected values: t1:  -17.500, t2:    2.500, t3:  -26.800
  t = 1       delta(I) =  ( 1, 1, 0, 0, 1, 1)
  t = 2       delta(I) =  ( 1, 1, 1, 1, 0, 0)
  t = 3       delta(I) =  ( 0, 0, 1, 1, 1, 1)


  Solution:
         objout   =  53726.00
         objin    =  28111.00
```

Also for HN EV approach, the sequence of closures is the same as in any WS solution. The problem of every HN is that we cannot change our mind after observation of randomness. Thus, if wait-and-see would find a better closures, EV approach will ignore this. But, take into account, the average of all WS solutions is a bit worse than our result for HN. This is caused by a small number of scenarios, and by fact that realisations are mostly symmetrical around the expected value with a big standard deviation.

HN Two-Stage with recourse. As shown in part 5.2, we can reformulate a stochastic program into its RF deterministic reformulation by relaxing constraints. This relaxation allows violation, which is penalised after the observation of randomness.

Same as before, we denote $\boldsymbol{\beta}_{ti}$ the direction of commodity $i \in \mathcal{C}$ in period $t \in T$, and consider simple recourse. We allow demand deviation from value $\mathbf{A}\mathbf{x}_{ti}$ by new variables y_{ti}^{+} and y_{ti}^{-} which are assigned after the realisation of random vector. Penalisation vector $\mathbf{q}_t^T = (q_{t+}, q_{t-})^T$ may be the same for each period, or it may change during time. Here, p_s^t denotes probability $P(\boldsymbol{\xi}_t = \boldsymbol{\xi}_t^s)$.

$$
\begin{array}{lll}
\min_{\boldsymbol{\delta}} & \displaystyle\sum_{t\in T}\left(\sum_{a\in A} c_a(f_{ta}(\delta_{ta}))f_{ta}(\delta_{ta}) + \sum_{s\in S} p_s^t \sum_{i\in\mathcal{C}} \mathbf{q}_t^T \mathbf{y}_{ti}(\boldsymbol{\xi}_t^s)\right) & \\
\text{s.t.} & f_{ta} = \displaystyle\sum_{i\in\mathcal{C}} x_{tia} & \forall a \in A, \forall t \in T, \\
& y_{ti}^{+}(\boldsymbol{\xi}_t^s)\boldsymbol{\beta}_{ti} - y_{ti}^{-}(\boldsymbol{\xi}_t^s)\boldsymbol{\beta}_{ti} = \mathbf{b}_{ti}(E\boldsymbol{\xi}_t) - \mathbf{A}\mathbf{x}_{ti} & \forall i \in \mathcal{C}, \forall t \in T, \\
& \mathbf{x}_{ti} \geq \mathbf{o} & \forall i \in \mathcal{C}, \forall t \in T, \\
& \mathbf{y}_{ti}(\boldsymbol{\xi}^s) \geq \mathbf{o} & \forall i \in \mathcal{C}, \forall t \in T, \\
& \sum_{a\in A}(1-\delta_{ta}) = R & \forall t \in T, \\
& f_{ta} \leq \delta_{ta} M & \forall a \in A, \forall t \in T, \\
& \delta_{ta} \in \{0,1\} & \forall a \in A, \forall t \in T. \\
& \boldsymbol{\delta}_t^T \boldsymbol{\delta}_r = |A| - 2R & \forall t, r \in T, t \neq r
\end{array}
\tag{6.11}
$$

Let assume constant penalisation in all periods, namely let $\mathbf{q}_t^T = (1000, 0)^T$ for each period $t \in T$. Hereby we mean "penalise only an overflow of the network." This approach should force the system to assume rather more than less drivers.

In such case, the total cost does not express the same value as in WS or HN EV. It is just "our valuation" of a flow. We present also realisations of random vectors and flow solution.

```
----------------------Traffic Assignment Problem: myTAP7 RF-------------
--------------------------------
Two-stage with recourse
---------------
              qplus     qminus
  t1          1000         0
  t2          1000         0
  t3          1000         0

  Optimal closures:
  t1          delta(I) =  ( 0, 0, 1, 1, 1, 1)
  t2          delta(I) =  ( 1, 1, 1, 1, 0, 0)
```

```
t3          delta(I) =  ( 1, 1, 0, 0, 1, 1)

Solution:
       objout   =  61006.00
       objin    =  37525.00

Randomness realisations:
scen 1     t1: -25     t2:  20     t3: -29
scen 2     t1:  -4     t2:   3     t3: -30
scen 3     t1: -13     t2:  20     t3: -36
scen 4     t1: -21     t2:  11     t3: -36
scen 5     t1: -21     t2: -15     t3: -22
scen 6     t1: -24     t2:   6     t3: -15
scen 7     t1: -20     t2: -14     t3: -33
scen 8     t1:  -4     t2: -10     t3: -20
scen 9     t1: -28     t2:   7     t3: -16
scen10     t1: -15     t2:  -3     t3: -31


Flow solution:
t1:
                 com1
       arc1        0
       arc2        0
       arc3       37
       arc4       37
       arc5       30
       arc6       30
t2:
                 com1
       arc1       43
       arc2       43
       arc3       44
       arc4       44
       arc5        0
       arc6        0
t3:
                 com1
       arc1       33
       arc2       33
       arc3        0
       arc4        0
       arc5       27
       arc6       27
```

Comments on SCP

Simplifications in SCP. In our assumptions, we simplified all models in many respects. One of this simplification was done by assuming the same probability distribution of number of drivers for every commodity. We conducted all computations with uniformly distributed volumes. Both these limitations are violated in real life - one street is blocked in the morning, while an other all day long. Deviations of the traffic are measured discretely, and distributions are given empirically, etc. Although all simplifications we made, in a real-life application the models we presented would change only in details.

Computational demands. For a large scale networks, we should be interested in increase of calculation efficiency. According to book [24], we may rewrite some conditions to achieve better computational times. For example, following nonlinear condition

$$\boldsymbol{\delta}_t^T \boldsymbol{\delta}_r = |A| - 2R \qquad \forall t, r \in T, t \neq r, \tag{6.12}$$

can be linearised at the cost of restrictions grow. This dot product represents sum of $|A|$ products of type $\delta_{ta}\delta_{ra}$. We replace each one of this members with a new binary variable γ_{ta} and constraints

$$\boldsymbol{\delta}_t + \boldsymbol{\delta}_r - \boldsymbol{\gamma}_t \leq \mathbf{1} \qquad \forall t, r \in T, t \neq r \tag{6.13}$$

$$\boldsymbol{\delta}_s - \boldsymbol{\gamma}_t \geq \mathbf{o} \qquad \forall s \in T \tag{6.14}$$

Equation 6.13 forces γ_{ta} to be 1 if δ_{ta} and δ_{ra} are both 1. Equation 6.14 forces γ_{ta} to be 0 if either δ_{ta} or δ_{ra} take the value 0. Then the condition 6.12 can be written as

$$\sum_{a \in A} \boldsymbol{\gamma}_{ta} = |A| - 2R \qquad \forall t \in T.$$

In models implemented in GAMS we did not use such linearisation, because of small instances of networks.

Conclusion

In this thesis, we described the well-known Minimal Cost Network Flow problem (MCF), which is generalised for multiple commodities. As a special case of this multi-commodity flow problems, we deal with TAP (Traffic Assignment Problem). TAP is a real-life application of network flow. It performs a modified variation of Multicommodity Flow problem which is nonlinear. This fact, and the linkage between TAP and game theory, make this pattern more complicated and complex.

As presented in section 4.3, there are two different points of view how to deal with such a problem as TAP. One insight is through network flow problems, which leads to best possible utilization of the system. This means, with respect to the objective of TAP, the average time driver needs to traverse a network is minimized. Because of the nature of individualities, drivers are not satisfied with such solution, and disturb the flow by their own decisions. We concentrate ourself on Beckmann's model that supposes cost function to be nondecreasing, continuous, nonnegative, and convex. Section 4.3 is focused on presenting differences between both approaches, and these are shown on a small instance of a network.

In the main part of the thesis, we focus on improvements of TAP, such as influence of randomness. We provide deterministic reformulations of given problems, where randomness appears in the cost functions or in the volume of drivers. In the first case, the results clearly show how good is the solution of here-and-now (HN) approach. Since wait-and-see (WS) approach is the best response for each realisation of random vector, naturally we compare HN with WS (see the table below). The comparison tool we use is price of anarchy ρ coefficient.

	$E\rho$	varρ
original task	1.01025	—
here-and-now	1.03706	0.00004
wait-and-see	1.01381	0.00014

We supposed symmetrical probability distribution with expected value equal to zero. The table shows us, how bad is the solution if we must decide before observing the realisation. WS exactly answers the observation, and under our assumptions, its expected price of anarchy converges to that from original problem. This is not the case of HN approach - price of anarchy is high with a low deviation (i.e. this is relatively precise expectation).

For a randomness in the right-hand-side vector, we use two-stage reformulation of the problem. Flow conservation is relaxed in order to achieve deterministic program. New variables are added, which express the violation of the constraint, and this is penalised. Because of the nature of problem, the lack of drivers in the network does not need to be penalised at all. We set this penalisation to be negative. Compare result observed solving task by different reformulations

	$E\rho$
original task	1.01025
here-and-now EV	1.01025
here-and-now TS with recourse	1.574751
wait-and-see	1.01250

Referring to the discussion of recourse model, we present just the first model with penalisation coefficients $q_+ = 70, q_- = -20$. A lower price of anarchy can be achieved by tightening penalisation conditions. HN approach shows the best expected result. This conclusion can be vague due to replacement the expected value by the its ceiling value.

In the last part, we shortly discuss result reached by Braess and Roughgarden, and we focus on Street cleaning problem. We define this task as problem of the best sequence of closing arcs in the network. This is formally shown to be a bilevel programming problem. After randomness is applied, we get results for various deterministic reformulations as follows

	total cost
wait-and-see average	53907
here-and-now EV	53726
here-and-now TS with recourse	61006

In fact, recourse reformulation corresponds to two-stage bilevel programming problem, which is as same as the other SCP task solvable in easier way. Results presented above show, that the average WS solution is worse than EV. This is not disturbing, since from the nature WS is the best response to each realisation of randomness. Such a deflection may be caused by small number of scenarios whose realisations lie symmetrically around the expected value, but with a relatively high standard deviation.

An interesting result is given by solution of two-stage HN reformulation. Here (see chapter 6), the optimal sequence of closures differ from WS- or EV-approach, respectively.

Although we present some transformations for linearising nonlinear constraints in SCP, computational demands may still remain very high. For a future research we recommend concentration on heuristic algorithms, which may achieve solution near to an optimal in much shorter time. After that, other additional restrictions may be supposed, and the computation time will not be so much affected.

References

[1] AKELLA, M. R. Unimodularity and Total Unimodularity. In: [online]. University of Buffalo, 2001, course materials [cit. 2012-05-01]. Dostupné z: `http://www.acsu.buffalo.edu/~nagi/courses/684/Unimodularity.pdf`

[2] BAZARAA, M. S., J. J. JARVIS a H. D. SHERALI. *Linear Programming and Network Flows.* 4th edition. Hoboken, New Jersey: John Wiley, 2010. ISBN 978-0-470-46272-0.

[3] BAZARAA, M. S., H. D. SHERALI a C. M. SHETTY. *Nonlinear Programming: theory and algorithms.* 3rd edition. Hoboken, New Jersey: John Wiley, 2006. ISBN 978-0-471-48600-8.

[4] BECKMANN, M., C. B. McGUIRE a C. B. WINSTEN. *Studies in the Economics of Transportation.* [online] Introduction by T.C. Koopmans. New Haven: Yale University Press, 1956. 232 s. 2011 [cit. 2012-05-01] Dostupné z: `http://cowles.econ.yale.edu/archive/reprints/specpub-BMW.pdf`

[5] BRAESS, D. Über ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung* [online]. 1968, č. 12, s 258-268 [cit. 2012-05-01]. Dostupné z: `http://homepage.ruhr-uni-bochum.de/dietrich.braess/#eng`

[6] CHUDAK, F. A., et al. *Static Traffic Assignment Problem. A comparison between Beckmann (1956) and Nesterov & de Palma(1998) models.* 7th Swiss Transport Research Conference. Monte Verita/Ascona, 2007.

[7] ČECHOVÁ, D. *Network Interdiction.* Brno, 2006. Diploma thesis. Brno University of Technology. Vedoucí práce Pavel Popela.

[8] DAFERMOS, S. C. a F. T. SPARROW. The Traffic Assignment Problem for a General Network. *Journal of Research of the National Bureau of Standards* [online]. National Bureau of Standards, 1969, 73B, č. 2, s. 91 [cit. 2012-05-01]. ISSN 0098-8979. Dostupné z: `http://cdm16009.contentdm.oclc.org/cdm/compoundobject/collection/p13011coll6/id/64392/rec/17`

[9] DANTZIG, G. B. a M. N. THAPA. *Linear programming: Introduction.* New York: Springer, 1997. ISBN 0-387-94833-3.

[10] DEMEL, J. *Grafy a jejich aplikace.* Vyd. 1. Praha: Academia, 2002, 257 s. ISBN 80-200-0990-6.

[11] DEMPE, S. *Foundations of Bilevel Programming.* Dordrecht: Kluwer Academic Publishers, 2002. Nonconvex Optimization and Its Applications, Vol. 61. ISBN 1-4020-0631-4

[12] FONIOK, J. Integer programming: course [online]. ETC Zurich, 2010, Chapter 8, 2012 [cit. 2012-05-01]. Dostupné z: `http://www.ifor.math.ethz.ch/teaching/lectures/integer_prog_ss10/`

[13] FUDENBERG, D. a J. TIROLE. *Game theory.* Cambridge: MIT Press, 1991, 579 s. ISBN 0-262-06141-4.

[14] GAMS. Solver Descriptions [online]. [cit. 2012-05-01]. Dostupné z: http://www.gams.com/solvers/solvers.htm

[15] HRABEC, D. *Modely stochastického programování pro inženýrský návrh.* Brno, 2011. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Pavel Popela.

[16] KALL, P. a S. W. WALLACE. *Stochastic Programming.* New York: Wiley, 1995, 307 s. ISBN 0-471-95158-7.

[17] MIKULÁŠEK, K. Grafy a algoritmy: studijní materiály. In: ÚM FSI VUT v Brně [online]. 2011 [cit. 2012-05-01]. Dostupné z: http://www.math.fme.vutbr.cz/default.aspx?catalog=3&catsrtext=11&catsrfield=38

[18] ROSENTHAL. *GAMS - A User´s Guide* [online]. 2008, 2008 [cit. 2012-05-01]. Dostupné z: http://www.gams.com/dd/docs/bigdocs/GAMSUsersGuide.pdf

[19] ROUGHGARDEN, T. Selfish Routing and the Price of Anarchy. In: [online]. Standford University, 2006 [cit. 2012-03-21]. Dostupné z: http://theory.stanford.edu/~tim/papers/optima.pdf

[20] SCHRIJVER, A. *Theory of linear and integer programming.* Chichester: Wiley, c1986, 471 s. ISBN 0-471-98232-6.

[21] WARDROP, J. G. *Some theoretical aspects of road traffic research.* Institute of Civil Engineers, 1952. Proceedings, Part II, Vol.1, 378 s.

[22] WASHBURN, A. a K. WOOD. Two-person zero-sum games for network interdiction. *Operations Research* [online]. 1995, Vol. 43, s. 243-251, 2003 [cit. 2012-05-01]. Dostupné z: http://faculty.nps.edu/kwood/docs/WashburnWood.pdf

[23] John Glen Wardrop. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012 [cit. 2012-05-01]. Dostupné z: http://en.wikipedia.org/wiki/John_Glen_Wardrop

[24] WILLIAMS, H. P. *Logic and Integer Programming.* 1st edition. New York: Springer, 2009. ISBN 978-0-387-92279-9.

[25] Fundamentals of Transportation/Route Chioce. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2012 [cit. 2012-05-01]. Dostupné z: http://en.wikibooks.org/wiki/Fundamentals_of_Transportation/Route_Choice#Example_1

[26] What if They Closed 42d Street and Nobody Noticed?. THE NEW YORK TIMES. [online]. [cit. 2012-05-01]. Dostupné z: http://www.nytimes.com/1990/12/25/health/what-if-they-closed-42d-street-and-nobody-noticed.html

List of used symbols

a, b, c	scalars, coefficients
x, y, w	variables, unknowns
A, B, C	sets
$\emptyset$	empty set
$\mathbb{N}, \mathbb{Z}, \mathbb{R}$	sets of positive integers, integers, and real numbers
$\mathbf{a}, \mathbf{b}, \mathbf{c}$	column vectors of coefficients
$\mathbf{x}, \mathbf{y}, \mathbf{w}$	column vectors of variables
$\mathbf{A}, \mathbf{B}, \mathbf{C}$	matrices $(a_{ij}), (b_{ij}), (c_{ij})$ of specified size
$\mathbf{I}$	identity matrix
$\mathbf{0}$	zero matrix of specified size
$\mathbf{o}$	column zero vector
$\det \mathbf{A}$	determinant of matrix $\mathbf{A}$
$G = (N, E)$	graph defined by set of nodes and set of edges
$G = (N, A)$	graph defined by set of nodes and set of arcs
s. t.	abbreviation for "subject to"
X, X^*	feasible set, set of optimal points
$\operatorname{argmin}_{\mathbf{X}}\{f(\mathbf{x})\|\mathbf{x} \in X\}$	set of optimal points (other notation)
$f^* = \min_{\mathbf{X}} f(\mathbf{x})$	optimal value of function f
z	objective function
$z^*, \mathbf{x}^*$	optimal value of objective function, optimal point (i.e. $x^* \in X^*$)
$\mathcal{C}$	set of commodities
$\mathbf{f}, f_a$	flow, flow on arc a
$\boldsymbol{\xi}, \boldsymbol{\xi}^s$	random vector and its realisation
Ξ	support of random vector $\boldsymbol{\xi}$
S	finite set of scenarios (support Ξ with finite cardinality)
$E\boldsymbol{\xi}$	expected value of random vector $\boldsymbol{\xi}$
$\operatorname{var}\boldsymbol{\xi}$	variance of random vector $\boldsymbol{\xi}$

A. Game Theory Background

This part presents some main ideas of game theory (see [13]), which are related to the concept of user equilibrium (UE), or Wardrop equilibrium, discussed in this thesis. The reader may compare perception of UE existence from different sides - from logical and native way it is just a conflict between decision makers, but from the game theoretical point of view, we may see deeper structure of the decision process.

We focus just on one part, where we consider games as *normal games of complete information*. Hereby, we presume all users (or *players*) have full knowledge about the game situation and about all possible decisions of other users. Moreover, all decision are made in the same time, or without knowing the actions of the other.

Definition A.1 (Normal-form representation game).
A normal-form game of n-players is defined by three elements: the set of players $\mathcal{P}$ (which we take to be finite set), the set of strategy space $\mathcal{S}_i$ of i-th player, and set of payoff function $u_i(\mathbf{s})$ of i-th player. The payoff function assigns to each combination $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ of strategies $s_i \in \mathcal{S}_i$ a real number.

Further, we refer to i-th player opponents by denotation $-i$ (i.e. all player except i-th). To avoid misunderstanding, the aim of each user is not to beat other users. Rather, each users' objective is to maximize his payoff.

Definition A.2 (Strictly dominated strategy).
In the normal-form game, let $s_i', s_i'' \in \mathcal{S}_i$ be strategies for player i. Strategy s_i' is *strictly dominated by strategy* s_i'' if, for each combination of the other players' strategies, it fulfils

$$u_i(s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_n) < u_i(s_1, \ldots, s_{i-1}, s_i'', s_{i+1}, \ldots, s_n)$$

Definition A.3 (Mixed strategy).
Suppose a game in normal-form, where $\mathcal{S}_i = \{s_{i1}, \ldots, s_{iK}\}$. The *mixed strategy* for player i is a probability distribution p_i over all his strategies, i.e. $p_i = (p_{i1}, \ldots, p_{iK})$ such that $0 \leq p_{ik} \leq 1$, for $k = 1, \ldots, K$, and $\sum_{k=1}^{K} p_{ik} = 1$.

A strategy $s_i \in \mathcal{S}_i$ is mostly called *pure strategy*. The meaning of a mixed strategy is to play a pure strategy with a given probability. Combination of mixed strategies $(p_1, \ldots, p_n)$ of all players is called *mixed strategy profile*, and we denote it simply by $\mathbf{p}$.

Definition A.4.
A mixed strategy profile $\mathbf{p}^*$ is a *Nash equilibrium* if, for all players i

$$u_i(p_i^*, p_{-i}^*) \geq u_i(s_i, p_{-i}^*) \qquad \forall s_i \in \mathcal{S}_i.$$

Thus at Nash equilibrium no player can only make his payoff worse by choosing other strategy. This equilibrium may be achieved by combination of pure or mixed strategies.

Prove of following theorem can be found for example in [13].

Theorem A.1 (Nash 1950).
In any finite normal-form game there exists at least one Nash equilibrium, possibly involving mixed strategies.

This key theorem exactly corresponds to user equilibrium used in this thesis. In an UE, deflecting from an optimal solution results in higher cost, which is the same principle as here. Its can be shown, that user equilibrium converges to Nash equilibrium for increasing number of users in a network.

B. GAMS

GAMS (General Algebraic Modeling System) is widely used optimization software, which solves various types of mathematical programs. It consists of a language compiler and a stable of integrated high-performance solvers. GAMS is specifically designed for modelling linear, nonlinear and mixed integer optimization problems. This chapter is based on [18]. We shortly describe the programming language, solver BARON used to solve nonlinear integer programs, and some properties to understand all the models used in the thesis. For this purpose, let us explain the model introduced in section 4.5.

Model Structure in Software GAMS

The basic components of GAMS model are

- `Sets`

  Declaration

  Assignment of members

- `Data`, i.e. `Parameters, Tables, Scalar`

  Declaration

  Assignment of values

- `Variables`

  Declaration

  Assignment of types

- Assignment of bounds and/or initial values (optional)

- `Equations`

  Declaration

  Definition

- `Model` and `Solve` statement

- `Display` statement (optional)

GAMS statements may be laid out typographically in almost any style, including multiple lines per statement, blank lines, and multiple statements per line. Each statement should be terminated with a semicolon. The GAMS compiler does not distinguish between upper- and lowercase letters.

Sets. In the code below, we declare three sets and gave them names I, N and J. This is followed by the label (optional) and enumeration of the set. Similar as in the further statements, we may declare one component using the keyword in singular, or multiple components using plural. Thus, we can write

```
sets     I set of arcs /arc1*arc7/
         N set of nodes /node1*node4/
         J set of commodities /com1,com2/;
```

or also

```
set  I set of arcs /arc1*arc7/ ;
set  N set of nodes /node1*node4/;
set  J set of commodities /com1,com2/ ;
```

Set J is given by its full enumeration, I contains all members $arc1, arc2, \ldots, arc7$. In models, that are concerned with interactions of elements within the same set, is useful to rename an already declared set. We may mark the set I by I'. This is served by the statement

```
Alias (I,Iprime) ;
```

Data. Data in GAMS are entered by list using `Parameters` statement, by `Tables`, or by direct assignment.

```
parameter FreeTime(I) free travel time of arc i
         /arc1 8, arc2 8, arc3 10, arc4 2, arc5 6, arc6 2, arc7 3/;
parameter Beta(I) multiplication coeficient of arc i
         /arc1 3, arc2 4, arc3 2, arc4 5, arc5 4, arc6 5, arc7 4/;
```

Here data are entered as indexed parameter $FreeTime(I)$, and values simply are listed. A particular assignment is separated by comma or newline. Zero is the default value of all parameters. A scalar is regarded as parameter without any domain. (Since section 4.5 does not contain declaration of scalar, let show an arbitrary example.)

```
scalar    T  average time  /100/ ;
```

The effect of a table is to declare a parameter, and to specify its domain as the set of ordered pairs. The values are given in this statement under the appropriate heading. If there are blank entries in the table, they are interpreted as zeros.

```
table A(N,I) incidence matrix
                 arc1    arc2    arc3    arc4    arc5    arc6    arc7
         node1    1       -1       1       1
         node2   -1        1                       1
         node3                    -1      -1               1      -1
         node4                                    -1      -1       1  ;
```

When data values are to be calculated, you first declare the parameter (i.e. give it a symbol and, optionally, index it), then give its algebraic formulation. GAMS will automatically make the calculations.

```
parameter     3Beta(I)  three times Beta-parameter ;
              3Beta(I) = Beta(I) * 3 ;
```

Variables. The decision variables must be declared within the `Variables` statement. Each variable is given a name, a domain if appropriate (never for objective function), and an optionally text.

```
variables        x(I,J)  flow through arc i of com. j
                 zOS     SO-total time cost
                 zUE     UE-total time cost
                 f(I)    total flow through arc i;
integer variable x;
```

Each variable must be assigned a type: Free, Positive, Negative, Binary, or Integer. The default is Free.

Equations. Format of the equation declaration is the same as for other GAMS entities. First comes the `Equations` statement, followed by the name and the domain, terminated with optional text.

```
equations        costSO        SO-objective function
                 flowCon(N,J)  flow conservation constraint
                 costUE        UE-objective function
                 flow(I)       flow through arc i;
```

Equations are defined as in the example below. First comes the name of the equation being defined, its domain, optionally the domain restrictions, and the symbol ".." of two dots. After that is written the expression: the left-hand side, in/equality relation, and the right-hand side. Variables can appear on arbitrary side of the equation, or both. GAMS uses these three relation symbols

$$\leq \ \ldots \ \texttt{=l=}$$
$$= \ \ldots \ \texttt{=e=}$$
$$\geq \ \ldots \ \texttt{=g=}$$

```
flow(I)..        f(I) =e= sum(J, x(I,J));
costSO..         zOS =e= sum(I,(FreeTime(I)+Beta(I)*f(I))*f(I));
costUE..         zUE =e= sum(I,(FreeTime(I)+Beta(I)*f(I)/2)*f(I));
flowCon(N,J)..   sum(I,A(N,I)*x(I,J)) =e= Demand(N,J);
```

Objective function. GAMS has no special entity by default referring to objective function. The function to be minimized is declared as scalar variable (variable without any domain), which is free.

Model and Solve statements. The statement `Model` in principle refers to a set of equations. Declaration of the model must contain its name and list of equations in slashes. For including all defined equations, we just type `/all/`

```
model tapOS /costSO, flow, flowCon/;
model tapUE /costUE, flow, flowCon/;
```

Once a model is declared, it can be solved with `solve` statement. The format of `solve` is following

- the keyword `solve`
- the name of the model to be solved
- the keyword `using`
- solver to use (`lp` fo linear programming, `nlp` for nonlinear prog., `mip` for mixed integer prog., `minlp` for mixed integer nonlinear prog.,etc.)
- the keyword `minimizing` or `maximizing`
- the name of the variable to be optimized

```
solve tapOS using minlp minimizing zOS ;
```

Display statement. The optimal values of variables can be displayed with the statement `display` followed by *variable*`.l`, *variable*`.m`, for primal or dual variables, respectively.

```
display x.l,zOS.l;
```

Solvers

BARON Due to typical TAP usage, we have to solve a special mathematical program. For this task we use in GAMS solver named BARON (The Branch-And-Reduce Optimization Navigator). BARON is solver for the global solution of nonlinear and mixed-integer programs , with no starting point needed.

BARON implements algorithms of the branch-and-bound type enhanced with a variety of constraint propagation and duality techniques for reducing ranges of variables in the course of the algorithm (see [14]). In addition to multiplication and division, BARON can handle as exponential, logarithmic, polynomial function, so function of absolute value, etc.

DICOPT Solver DICOPT uses a standard GAMS MIP and NLP solvers to solve the MIP and NLP subproblems generated by the algorithm. This means that in order to use DICOPT we need to use a MIP solver and a NLP solver.

ALPHAECP ALPHAECP is a solver based on the extended cutting plane method. The solver can be applied to general mixed integer nonlinear programming problems and global optimal solutions can be ensured for pseudo-convex MINLP problems.

C. What is on CD

All presented problems have been implemented in software GAMS (see appendix B). Each folder contains an output file, usually called *OUT.txt*, which summarizes all results. The appropriate folders containing also code in GAMS are included to this thesis in following order:

- *Jan_Holesovsky_Masters_Thesis_2012.pdf*

  This thesis in PDF format.

- *myTAP1*

  Implementation of "One simple model" from page 33.

- *myTAP2*

  Implementation of wait-and-see and here-and-now deterministic reformulation of stochastic programming problem with random vector in objective function (see subsection 5.2).

- *myTAP3*

  Implementation of wait-and-see and here-and-now deterministic reformulation of stochastic programming problem with random vector in right-hand-side vector (see subsection 5.2).

- *myTAP4*

  Stochastic program with randomness in constraint solved as two-stage problem with recourse.

- *myTAP5*

  Implementation of the R-closure problem from page 55.

- *myTAP5 enumeration*

  The best R-closure problem solved by full enumeration. Folder contains GAMS file *myTAP5 enumer* and output in file *OUT.txt*.

- *myTAP scp*

  Implementation of street cleaning problem (see page 60).

- *myTAP scp enumeration*

  Implementation of street cleaning problem solved by full enumeration.

- *myTAP 7*

  Wait-and-see and here-and-now expected value approaches applied on street cleaning problem.

- *myTAP7 RF*

  Two-stage deterministic reformulation of stochastic SCP with recourse.